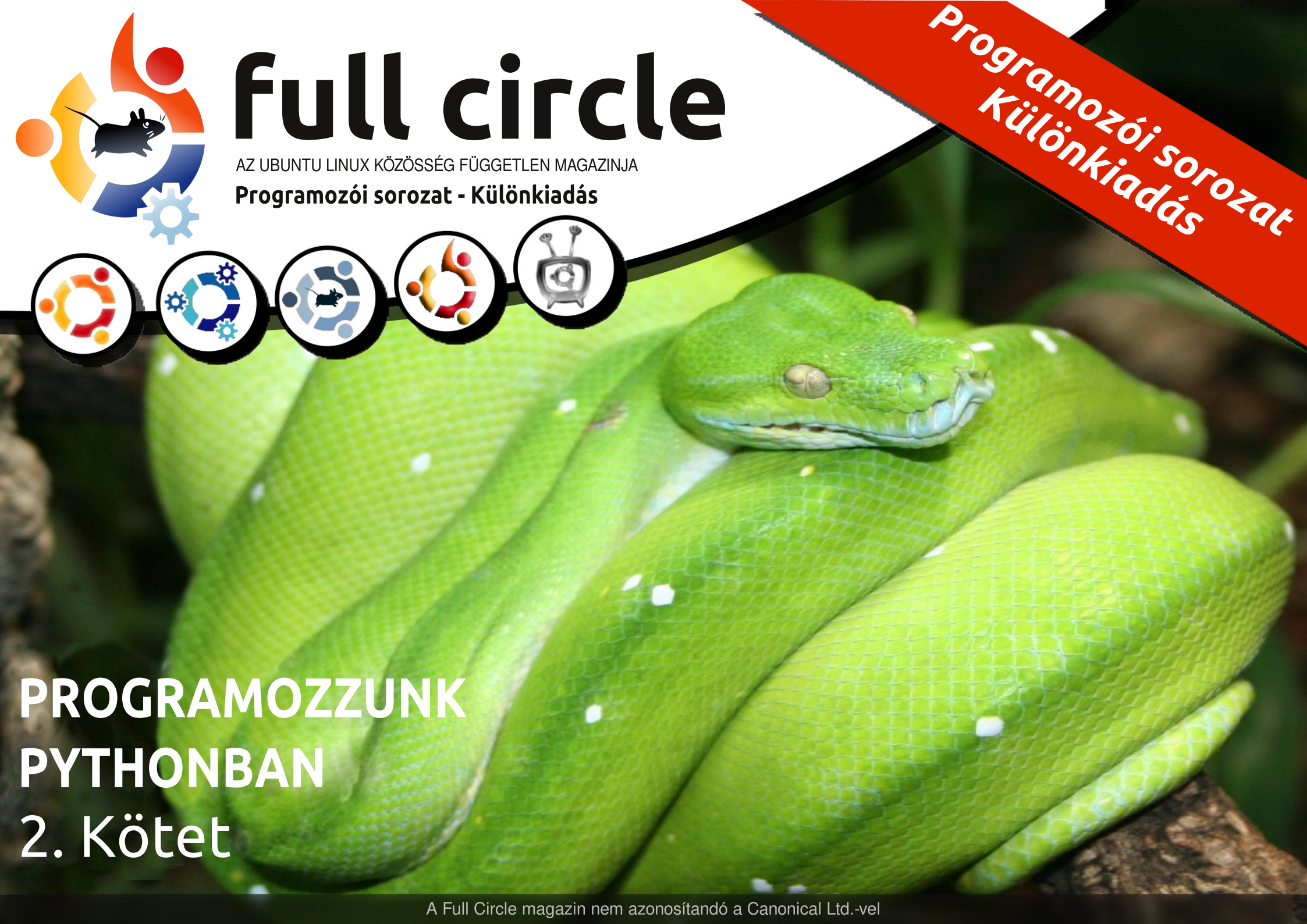
full circle

AZ UBUNTU LINUX KÖZÖSSÉG FÜGGETLEN MAGAZINJA

Programozói sorozat - Különkiadás

Programozói sorozat Különkiadás



PROGRAMOZZUNK PYTHONBAN

2. 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” 9.-16. részének** az újabb kiadása (a magazin 35.-42. 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!

Hogyanok

Programozzunk Pythonban - 9. rész

Írta: Greg Walters

Előző részek:
FCM 27.-34. számokban
az 1.-8. részek

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 számítógépeden te is mp3 formátumban tárolod kedvenc zenéidet. Amikor kevesebb, mint 1000 számod van, akkor még elég könnyű megjegyezned, hogy mi és hol van. Mivel régebben DJ voltam és zenéim nagy részét átkonvertáltam, ezért nekem ennél jóval több MP3-am van. Régebben a legnagyobb problémát a merevlemez kapacitása jelentette számomra. Azonban manapság a legfrusztrálóbb dolog, hogy nem tudom észben tartani, hogy mim és hol van.

Ebben a leckében megnézzük, hogy hogyan tudunk létrehozni az MP3-ainknak egy katalógust. Megismerkedünk néhány újabb python koncepcióval, amellett, hogy felfrissítjük az adatbázisokról szerzett tudásunkat is.

Először, nem árt tudnunk arról, hogy egy MP3 fájl tartalmazhat önmagáról információkat. Ilyen adat a dal címe, albuma, szerzője, stb. Ezek az információk az ID3 elemekben helyezkednek el, amit meta-adatnak is szoktunk hívni. Az első időkben egy MP3 fájlban csak igen korlátozott információmennyiség volt eltárolható. Ez eredetileg a fájl legutolsó 128 bájtjában foglalt helyet. Mivel ennek a blokknak igen kicsi volt a mérete, ezért csak 30 karakter hosszú lehetett a dal címe, szerzőjének neve, stb. Sok zenefájl számára ennyi éppen elegendő volt, de (és ez az egyik örök kedvenc számom) ha például a „Clowns (The Demise of the European Circus with No Thanks to Fellini)” számról volt szó, akkor csak az első 30 karakter volt meg a címből. Ez NAGYON frusztrálólag hatott igen sok emberre. A „sztenderd” ID3-as elem egy idő után ID3v1-ként lett ismert, miután bevezettek egy új formátumot, amit – meglepő módon - ID3v2-nek neveztek el. Ez az új formátum lehetővé tette változó hosszúságú adatok tárolását a fájl elején, amellett, hogy a régi ID3v1 meta-adat - a régebbi típusú lejátszók miatt - még mindig helyet kapott a fájl végén. Most már 256 Mb-nyi meta-adatot is eltárolhattunk. Ez pont ideális volt a rádióállomások és az olyan őrültek számára, mint amilyen én is vagyok. Az ID3v2 rendszer alatt minden egyes adatcsoport egy frame-ben (szelet) kap helyet, és mindegyik ilyen frame-nek van egy azonosítója. A korai ID3v2-ben ez az azonosító három karakter hosszú volt. A jelenlegi verzió (ID3v2.4) már négy karakterest használ.

Régen egyszerűen bináris olvasás módban megnyitottuk volna a fájlt, és addig kutakodtunk volna benne, amíg meg nem találtuk a keresett információt. Mivel nem voltak sztenderd könyvtáraink, amik ezt lekezelték volna, ezért ezt igen sok munka volt megcsinálni. Szerencsére, mostanra ezek már rendelkezésünkre állnak. Mi itt a Mutagen nevű projektet fogjuk használni. Menjünk is a Synapticba és telepítsük a python-mutagent. Ha szeretnénk, akkor itt egy keresést is lefuttathatunk az „ID3”-ra. Azt fogjuk tapasztalni, hogy több mint 90 csomagot dob ki eredményül (Karmicban), majd, ha a „Python” szót is begépeljük a gyorskereső mezőbe, akkor nyolc csomagot találunk. Mindegyik mellett szólnak érvek és ellenérvek, de mostani projektünkhöz a Mutagent fogjuk választani. Nyugodtan beleáshatjuk magunkat a többibe is kiegészítő tanulás

céljából.

Most, hogy már fel van telepítve a Mutagen, elkezdhetünk kódolni.

Induljunk egy „mCat” nevű projekt készítésével. Ezután jöhetnek az importjaink.

```
from mutagen.mp3 import MP3

import os

from os.path import
join,getsize,exists

import sys

import apsw
```

Ezekkel többnyire már találkoztunk. Ezt követően létre szeretnénk hozni a függvény deklarációinkat.

```
def MakeDataBase():
    pass
def S2HMS(t):
    pass
def WalkThePath(musicpath):
    pass
def error(message):
    pass
def main():
    pass
def usage():
    pass
```

ÁÁÁ... itt valami új dolog van. Készen vagyunk a main és a usage metódusokkal. Mire lesznek ezek jók? Mielőtt megbeszélnénk őket, helyezzünk el még egy dolgot.

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

Mi a fene ez? Ezzel a trükkel azt érjük el, hogy a fájlunkat képesek leszünk különálló alkalmazásként, illetve újrahasználható modulként egy másik programba beimportálva is használni. Gyakorlatilag azt mondja ki, hogy „HA ez a fájl a fő alkalmazás, akkor a main rutin meghívásával tudjuk futtatni, különben közvetlenül egy kiegészítő modulként fogunk hivatkozni a metódusaira egy másik programból”.

Ezután, fogalmazzuk meg a usage függvényt. Lent látható a usage rutin teljes kódja.

Itt fogjuk létrehozni a felhasználó számára megjelenő üzenetet, mely akkor jelenik meg, amikor nem megfelelő paraméterekkel indítják el a programunkat. Figyeljük meg, hogy a „\n” kifejezést egy új sor, a „\t”-t egy tab kiíratásához használjuk. Ezeken felül használjuk még a „%s”-t az alkalmazás nevének bekéréséhez, mely a sys.argv[0]-ban van eltárolva. Ezután meghívjuk az error rutint a „message” kiíratásához, majd a sys.exit(1)-el kilépünk az alkalmazásból.

Ha ezzel készen vagyunk, akkor készítsük el az error rutint. Itt van a teljes kódja:

```
def error(message):
    print >> sys.stderr,
str(message)
```

Egy átirányításnak nevezett dolgot alkalmazunk (a „>>”). Amikor a print-et hívjuk, valójában arra utasítjuk a Pythont, hogy írassa vagy streamelje ki a sztenderd kimenetre a megadott paramétereket, ami legtöbbször az éppen futó terminált jelenti. Ennek eléréséhez (a színfalak mögött) az stdoutot használjuk. Amikor hibaüzenetekkel dolgozunk, akkor az

```
def usage():
    message = (
       '=============================================\n'
       'mCat - Finds all *.mp3 files in a given folder (and sub-folders),\n'
       '\tread the id3 tags, and write that information to a SQLite database.\n\n'
       'Usage:\n'
       '\t{0} <foldername>\n'
       '\t WHERE <foldername> is the path to your MP3 files.\n\n'
       'Author: Greg Walters\n'
       'For Full Circle Magazine\n'
       '=============================================\n'
       ).format(sys.argv[0])
    error(message)
    sys.exit(1)
```

```
def main():
    global connection
    global cursor
    #-----------------------------------------------
    if len(sys.argv) != 2:
        usage()
    else:
        StartFolder = sys.argv[1]
        if not exists(StartFolder): # From os.path
            print('Path {0} does not seem to
exist...Exiting.').format(StartFolder)
            sys.exit(1)
        else:
            print('About to work {0}
folder(s):').format(StartFolder)
        # Create the connection and cursor.
        connection=apsw.Connection("mCat.db3")
        cursor=connection.cursor()
        # Make the database if it doesn't exist...
        MakeDataBase()
        # Do the actual work...
        WalkThePath(StartFolder)
        # Close the cursor and connection...
        cursor.close()
        connection.close()
        # Let us know we are finished...
        print("FINISHED!")
```

stderrt szoktuk megadni. Mi is ezt tesszük: átirányítjuk a print kimenetét az stderr folyamra.

Most dolgozzunk egy kicsit a main függvényen. Itt fogjuk beállítani az adatbázis kapcsolatot és kurzort, majd vetünk egy pillantást a parancssori paraméterekre. Ha minden klappol, akkor meghívjuk a saját függvényeinket az igazi munka elvégzéséhez. A kód fentebb látható.

Mint ahogy múltkor, most is létrehozunk két globális változót, melyeket az adatbáziskezeléshez használunk. Ezeket connection-nek és cursor-nak hívjuk. Ezután megnézzük a parancssorból kapott paramétereket (ha vannak egyáltalán). Mindezt a sys.argv utasítással tesszük. Két paramétert keresünk: az első az alkalmazás neve, a második az MP3 fájlok elérési útja. Ha nincs meg mindkét paraméter, akkor meghívjuk a usage szubrutint, mely kiíratja az előbb megírt üzenetet és kilép. Ha viszont megkaptuk mindkettőt, akkor az If-ünk else ágára ugrik a vezérlés. Ezt követően eltároljuk a kiinduló útvonalat a StartFolder változóban. Arra azonban ügyelnünk kell, hogy ha szóköz van az elérési útban - mint például az (/mnt/musicmain /Adult Contemporary)-ben - a space után következő karakterek egy új paraméterként fognak értelmeződni. Ezért ha bármikor szóköz van az elérési útban, akkor ne felejtsük el az egészet idézőjelek közé rakni. Következőnek beállítjuk a connectiont és a cursort, elkészítjük az adatbázist, majd jön a tényleges munka, amikor is meghívjuk a WalkThePath rutint, majd lezárjuk a kapcsolatot egy üzenet kíséretében, ami tudatja a felhasználóval, hogy készen vagyunk. Itt van a WalkThePath teljes kódja: http://pastebin.com/CegsAXjW.

Először kinullázzuk azt a három számlálót, melyekben a munka állapotát fogjuk követni. Ezután megnyitunk egy fájlt, melyben az esetlegesen felmerülő hibákat fogjuk loggolni. Ezt követően, rekurzívan végigmegyünk a felhasználó által megadott útvonalon. Gyakorlatilag a megadott útvonalban lévő mappákba lépegetünk be és ki, miközben bármilyen .mp3 kiterjesztésű fájl után kutatunk. Ha ezzel készen vagyunk, akkor megnöveljük a mappa és a fájl számlálóját, hogy tudjuk menynyi fájlon vagyunk már túl. Ha végeztünk, akkor átmegyünk minden fájlon. Kinullázunk minden helyi változót, mely az adott számról tartalmaz információt. Itt használjuk az os.pathban lévő „join” függvényt, melyet a megfelelő elérési út és fájlnév létrehozásához használunk. Ezt fogja megkapni a mutagen, amiből meg tudja állapítani, hogy hol keresse az adott fájlt. Ezt követően átadjuk a fájlnevet az MP3 osztálynak melyért az „audio” egy példányát kapjuk cserébe.

Ha elkészültünk, akkor kiszedjük a fájlból az összes ID3 elemet és végiglépkedünk a listán, olyan tagok után kutatva, amelyekre szükségünk van, majd ezeket az ideiglenes változókhoz rendeljük. Ezzel a módszerrel minimalizáljuk a lehetséges hibák számát. Vessünk egy pillantást a kód azon részére, mely a számok sorszámát tárolja. Amikor ezt visszakapjuk a mutagentől, akkor az lehet egy szám, egy „4/8”-hoz hasonló sztring vagy „_trk[0]” és „_trk[1]”, vagy akár semmi is. A try/except vezérlési szerkezetet alkalmazzuk az olyan hibák elkapásához, melyek emiatt előjöhetnek. Most nézzük meg a rekordok kiírását. Egy kicsit máshogy járunk el mint múltkor. Most is létrehozzuk az SQL utasítást, mint eddig, de a behelyettesítendő értékek helyére a „?”-et írjuk. Az ASPW webhelye szerint ez egy biztonságosabb módszer. Nem fogok velük vitatkozni. A végén lekezelünk minden előforduló hibát. Legtöbbször ezek a „TypeError”-ok vagy „ValueError”-ok lesznek, melyeket a nem lekezelhető Unicode karakterek okoznak. Álljunk meg egy pillanatra a sztring érdekes formázása és kiíratása fölött. Itt nem használjuk a „%” helyettesítő karaktert. Helyette a „{0}” típus helyettesítést alkalmazzuk, ami a Python 3.x specifikáció része. Az alapvető alakja az alábbi:

```
Print('String that will be printed with {0} number of statements").format(replacement values)
```

Ezt az alapvető szintaxist használjuk az „efile.writelines”-nál is.

Végül nézzük meg az S2HMS metódust. Ez a rutin a szám hosszát kapja paraméterként, amely egy mutagen által visszaadott lebegőpontos szám. Ezt fogjuk „Óra:Perc:Másodperc” vagy „Perc:Másodperc” alakra átkonvertálni. Figyeljük meg a return utasítást. Ismét a Python 3.x-ben bevezetett szintaxist alkalmazzuk. Van azonban egy teljesen új dolog is. Helyettesítési halmazokat használunk (0, 1, és 3), de mi a „:02n” az 1-es és 2-es szám után? Ez azt mondja ki, hogy két helyen szeretnénk csak bevezető nullákat látni. Tehát, ha a dal mérete 2 perc 4 másodperc, akkor a visszaadott karakterlánc „2:04” lesz és nem „2:4”.

A programunk teljes kódja itt található: http://pastebin.com/rFf4Gm7E.

Ez minden mára. Egy kicsit rövidebb lett mint múltkor, de talán picit bonyolultabb is. Ha van egy kis időnk, akkor nyugodtan belevethetjük magunkat a Mutagenbe, hátha találunk még valami érdekeset. Biztosan örömmel fogjuk venni, hogy MP3 fájlok kezelésénél többre is képes.

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

Programozzunk Pythonban - 10. rész

Írta: Greg Walters

Előző részek:
FCM 27.-35. számokban
az 1.-9. részek

Itt használható:

ubuntu kubuntu xubuntu

Kategóriák:

Fejlesztés Grafika Internet M/média Rendszer

Eszközök:

CD/DVD Merevlemez USB eszköz Laptop Vezeték nélküli

Valószínűleg mindenki hallotta már az XML kifejezést. Viszont nem biztos, hogy tudjuk, mi is ez. E havi anyagunk témája az XML lesz. Célunk, hogy

- megismerjük az XML-t
- megmutassuk, miként kell írni és olvasni XML fájlokat saját alkalmazásokban.
- felkészüljünk a következő alkalomra tartogatott viszonylag nagy XML projektre.

Nos... Beszéljünk az XML-ről. A HTML-hez hasonlóan, az XML is egy mozaikszó, ami az eXtensible Markup Language rövidítése. Az adatok hatékony tárolására és interneten, vagy más kommunikációs hálózaton való továbbítására találták ki. Gyakorlatilag az XML egy szöveges fájl, ami a saját „tag”-jaink (avagy elemeink) által van formázva, így elég öndokumentáló a felépítése. Mivel egy szöveges állományról van szó, össze lehet tömöríteni a gyorsabb és könnyebb továbbításhoz. A HTML-lel ellentétben az XML önmagában nem képes semmire. Nem foglalkozik azzal, hogy te hogyan akarod az adataidat elrendezni. Ahogy az imént mondtam, XML írásakor nem kell egy sor általános „tag”-ra hagyatkozni. Elkészíthetjük a sajátjainkat is.

Nézzünk egy általános példát XML fájlra:

```
<root>
     <node1>Data
Here</node1>
     <node2
attribute="something">Node 2
data</node2>
     <node3>
        <node3sub1>more
data</node3sub1>
     </node3>
</root>
```

Az első feltűnő dolog az indentálás. Gyakorlatilag csak az emberi fogyasztás megkönnyítése miatt van. Az XML ugyanolyan jól lenne így is:

```
<root><node1>Data
Here</node1><node2
attribute="something">Node 2
data</node2><node3><node3sub
1>more
data</node3sub1></node3></ro
ot>
```

A „<>” csúcsos zárójelek között lévő tagokra van néhány előírás. Először is, csak egyetlen szóból állhatnak. Másodszor, amikor van egy nyitó tagod (pl. <root>), akkor kell valahol lennie egy ugyanolyan záró tagnak is. A záró elem egy „/” jellel kezdődik. A tagok is méretérzékenyek: a <node>, <Node> és <NodE> mind különböző elemet jelöl, amikhez megegyező zárótagnak KELL tartoznia. Az elemnevek tartalmazhatnak betűket, számokat és egyéb karaktereket is, de nem kezdődhetnek számmal vagy központozással. A „-”, „.” és „:” jeleket a nevekben lehetőleg kerüljük el, mivel néhány alkalmazás parancsnak, vagy objektum adattagnak tekintheti őket. Továbbá a kettőspontoknak is valami más szerepük van.

Minden XML fájl alapvetően egy fa is - a gyökértől kiindulva szétágazik. Minden XML állománynak tartalmaznia KELL egy gyökér elemet, ami minden más elemnek az őse. Vessünk ismét egy pillantást a példánkra. A root alatt három gyermek elemünk van (node1, node2, node3). A root elem leszármazottai közti kapcsolathoz hasonlóan, a node3 is a node3sub1-nek a szülője.

Figyeljük meg a node2-őt. Vegyük észre, hogy a szokásos, elemek közötti adatok mellett van még egy, tulajdonságnak (attribute) nevezett értéke is.

Manapság igen sok fejlesztő azonban elkerüli használatukat, mivel az elemek közötti értékként is épp elég hatékonyak lesznek, ezen felül kevesebb pepecseléssel is jár nélkülözésük. Ennek ellenére látni fogjuk, hogy még mindig használják őket. Egy kicsit később még kitérünk rájuk.

Nézzünk most meg egy hasznos példát (balra lent).

Itt a gyökérelem a „people” (emberek), melynek két gyermeke van, „person” (személy) néven. Minden personnek van hat leszármazottja: firstname (keresztnév), lastname (vezetéknév), gender (nem), address (cím), city (város) és state (állam). Első pillantásra azt hinné az ember, hogy ez egy adatbázis (gondoljunk vissza az utóbbi néhány cikkre), és nem is tévednénk nagyot. Ami azt illeti, néhány alkalmazás XML fájlokat használ egyszerű adatbáziskezelésre is. Az XML-eket olvasó alkalmazások megírása viszonylag egyszerűen megoldható. Csak nyissuk meg a fájlt, olvassunk be minden sort, majd az elem típusa szerint foglalkozzunk a beolvasott sorokkal, végül zárjuk le a fájlt. Mindazonáltal ennél vannak kifinomultabb megoldások is.

```
<people>
    <person>
        <firstname>Samantha</firstname>
        <lastname>Pharoh</lastname>
        <gender>Female</gender>
        <address>123 Main St.</address>
        <city>Denver</city>
        <state>Colorado</state>
    </person>
    <person>
        <firstname>Steve</firstname>
        <lastname>Levon</lastname>
        <gender>Male</gender>
        <address>332120 Arapahoe Blvd.</address>
        <city>Denver</city>
        <state>Colorado</state>
    </person>
</people>
```

A soron következő példákban egy ElementTree-nek nevezett könyvtár modult fogunk használni. A Synapticból közvetlenül fel lehet telepíteni a python-elementtree csomaggal. Én viszont inkább az ElementTree weblapjáról töltöttem le a forrást (elementtree-1.2.6-20050316.tar.gz): (http://effbot.org/downloads/#elementtree). Amint lejött, a csomagkezelővel kitömörítettem egy ideiglenes mappába. Miután ebbe beleléptem, végrehajtottam a „sudo python setup.py install” utasítást. Ezzel a fájlok bemásolódtak a python közös mappájába, hogy python 2.5-ből, vagy 2.6-ból használni lehessen őket. Most már elkezdhetünk dolgozni. Hozzunk létre egy új mappát ehavi kódunknak, és kedvenc szövegszerkesztőnket használva, másoljuk ebbe a fenti XML adatot, majd mentsük el „xmlsample1.xml" néven.

Ami a kódunkat illeti, az első dolog, amit meg kell tennünk, hogy leellenőrizzük az ElementTree telepítést. Itt a kód:

```
import
elementtree.ElementTree as ET

tree =
ET.parse('xmlsample1.xml')

ET.dump(tree)
```

Amikor a tesztprogramot futtatjuk, valami ilyesmit kellene kapnunk:

```
/usr/bin/python -u
"/home/greg/Documents/articles/xml/read
er1.py"

<people>
    <person>
        <firstname>Samantha</firstname>
        <lastname>Pharoh</lastname>
        <gender>Female</gender>
        <address>123 Main St.</address>
        <city>Denver</city>
        <state>Colorado</state>
    </person>
    <person>
        <firstname>Steve</firstname>
        <lastname>Levon</lastname>
        <gender>Male</gender>
        <address>332120 Arapahoe
Blvd.</address>
        <city>Denver</city>
        <state>Colorado</state>
    </person>
</people>
```

Mindössze annyit csináltunk, hogy megnyitottuk az ElementTree-vel a fájlt, alapelemekre bontottuk a tartalmát, végül kiírattuk úgy, ahogy az a memóriában van. Semmi különös.

Most cseréljük le az előbbi kódot a következőre:

```
import
elementtree.ElementTree as ET

tree =
ET.parse('xmlsample1.xml')

person =
tree.findall('.//person')

for p in person:
    for dat in p:
        print "Element: %s -
Data: %s" %(dat.tag,dat.text)
```

és futtassuk újra. Most ilyen kimenetet kellene kapjunk:

```
/usr/bin/python -u
"/home/greg/Documents/article
s/xml/reader1.py"

Element: firstname - Data:
Samantha
Element: lastname - Data:
Pharoh
Element: gender - Data:
Female
Element: address - Data: 123
Main St.
Element: city - Data: Denver
Element: state - Data:
Colorado
Element: firstname - Data:
Steve
Element: lastname - Data:
Levon
Element: gender - Data: Male
Element: address - Data:
332120 Arapahoe Blvd.
Element: city - Data: Denver
Element: state - Data:
Colorado
```

Most már minden adat a tag neve mellé került. Egyszerűen csak ki kellett íratnunk az eredményt. Nézzük meg, hogy mink van itt. Az ElementTree szétbontotta a fájlt egy tree nevű objektumba. Ezután megkértük, hogy keresse meg a person összes példányát. A használt példában csak kettő ilyen van, de ez a szám lehet egy, vagy akár ezer is. A person a people leszármazottja, és tudjuk, hogy a people egyszerűen egy gyökérelem. Az összes adatunk a personba van elhelyezve. Létrehozunk egy for ciklust az összes person objektumon való átlépegetésre. Ezután készítünk egy másik for ciklust az adatok persononkénti kiszedéséhez, majd megjelenítjük az elem nevét (.tag) és adatát (.text).

Egy való életből vett példa: én és a családom egy Geocaching nevezetű tevékenységben veszünk részt. Azoknak, akik esetleg nem tudnák, mi is ez: egy „kocka” kincsvadászat, ami kézi GPS eszközöket használ valaki más által elrejtett dolgok felkutatásához. Általános koordinátákat egy weblapon helyezik el, néha egy-két nyom kíséretében. Nekünk annyi a dolgunk, hogy bepötyögjük a koordinátákat és megpróbáljuk megkeresni. A Wikipedia szerint több mint 1.000.000 aktív geocache weblap van világszerte, köztük valószínűleg akad egy a környékeden is. Én két weblapot használok célpontok kereséséhez. Az egyik a http://www.geocaching.com/ a másik pedig a http://navicache.com/. Vannak még mások is, de talán ez a kettő a legnagyobb.

Mindegyik célpont információja egy alap XML fájlban van eltárolva. Vannak olyan alkal-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<loc version="1.0" src="NaviCache">
   <waypoint>
      <name id="N02CAC"><![CDATA[Take Goofy Pictures at Grapevine Lake  by g_phillips
Open Cache: Unrestricted
Cache Type: Normal
Cache Size: Normal
Difficulty: 1.5
Terrain   : 2.0]]></name>
      <coord  lat="32.9890166666667"  lon="-97.0728833333333" />
      <type>Geocache</type>
      <link text="Cache Details">http://www.navicache.com/cgi-
bin/db/displaycache2.pl?CacheID=11436</link>
   </waypoint>
</loc>
```

Navicache fájl

mazások, melyek be tudják olvasni és átküldeni az adatokat a GPS eszközre. Néhányuk adatbázis-kezelő programként viselkedik - lehetőség van figyelemmel kísérni tevékenységünket (néha) térképek segítségével. Most még csak a letöltött fájl szétbontására fogunk koncentrálni.

A Navicachen találtam egy nem túl régi texasi feladványt. Az előző oldal alján látható.

Másoljuk ki a fenti adatokat és mentsük el „Cache.loc” névvel. Mielőtt nekiállnánk kódolni, vizsgáljuk meg a rejtvény fájlját.

Az első sor csak annyit mond nekünk, hogy ez egy validált XML fájl. Ezt nyugodtan figyelmen kívül hagyhatjuk. A következő sor (ami a „loc”-kal kezdődik) a gyökér, melynek version (verziószám) és src (forrás) tulajdonságai vannak. Emlékszünk még, amikor korábban azt mondtam, hogy néhány fájlban találkozhatunk attribútumokkal? Lesznek még mások is ebben az állományban, ahogy haladunk. Még egyszer elmondom, hogy a gyökér ebben az esetben figyelmen kívül hagyható. A következő sor a waypoint (csomópont) gyermekelemet tartalmazza. (A csomópont ebben az esetben a rejtekhely helyét jelenti.) Ez egy számunkra fontos adat lesz. Itt van a rejtekhely neve, illetve szélességi és hosszúsági koordinátái, a zsákmány típusa és egy további információkat tartalmazó honlap linkje. A name (név) elem egy hosszú karakterlánc, amiben van egy rakat hasznos információ, de ehhez előbb fel kell bontanunk magunknak. Készítsünk el egy új alkalmazást, ami beolvassa és megjeleníti ezt a fájlt. Legyen a neve „readacache.py”. Kezdjük az előző példabeli import és parse utasításokkal.

```
import
elementtree.ElementTree as ET

tree = ET.parse('Cache.loc')
```

Egyelőre szeretnénk kiolvasni a waypoint tagban lévő adatokat. Ehhez az ElementTree .find függvényét használjuk. Az eredményt a „w” objektumban kapjuk vissza.

```
w = tree.find('.//waypoint')
```

Ezután át szeretnénk futni az összes adaton. Egy for ciklust fogunk ehhez használni. A ciklusban meg fogjuk nézni, hogy a tag a „name”, „coord”, „type” és „link” közül melyik. Ettől függően fogjuk kiszedni a benne lévő adatokat későbbi kiíratásra.

```
for w1 in w:
    if w1.tag == "name":
```

Mivel először a „name” tagot keressük, nem árt, ha megnézzük a beolvasandó adatokat.

```
<name
id="N02CAC"><![CDATA[Take
Goofy Pictures at Grapevine
Lake  by g_phillips

Open Cache: Unrestricted

Cache Type: Normal

Cache Size: Normal

Difficulty: 1.5

Terrain   : 2.0]]></name>
```

Ez egy igen hosszú sztring. A rejtekhely „id” (azonosító) egy tulajdonság. A név a „CDATA” és „Open Cache:” közötti rész. Fel fogjuk darabolni ezt a sztringet további, nekünk megfelelő részekre. Egy karakterlánc részét az alábbi módon kapjuk meg:

```
newstring =
oldstring[startposition:endpo
sition]
```

Tehát, a lenti kóddal kiszedhetjük a nekünk kellő részeket.

Ezután meg kell szereznünk azt az azonosítót, ami a name

```
# Get text of cache name up to the phrase "Open Cache: "
CacheName = w1.text[:w1.text.find("Open Cache: ")-1]
# Get the text between "Open Cache: " and "Cache Type: "
OpenCache = w1.text[w1.text.find("Open Cache:
")+12:w1.text.find("Cache Type: ")-1]
# More of the same
CacheType = w1.text[w1.text.find("Cache Type:
")+12:w1.text.find("Cache Size: ")-1]
CacheSize = w1.text[w1.text.find("Cache Size:
")+12:w1.text.find("Difficulty: ")-1]
Difficulty= w1.text[w1.text.find("Difficulty:
")+12:w1.text.find("Terrain   : ")-1]
Terrain =   w1.text[w1.text.find("Terrain   : ")+12:]
```

tag tulajdonságában van. Attribútumok jelenlétét így kérdezzük le (természetesen most tudjuk, hogy mik vannak):

```
if w1.keys():
   for name,value in
w1.items():
      if name == 'id':
         CacheID = value
```

Foglalkozzunk most a maradék - koordináták, típus és link - elemekkel, a kód itt középen látható:

Végül kiíratjuk a lentebb látható módon.

Már most eleget tudunk ahhoz, hogy a legtöbb XML fájlt olvashassuk. Mint máskor is, a teljes forráskódot a honlapomról lehet letölteni.

Következő alkalommal felhasználjuk megszerzett XML tudásunkat egy csodálatos időjárásjelentő oldal lekérdezéséhez és terminálban való megjelenítéséhez.

Jó szórakozást!

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

```
elif w1.tag == "coord":
   if w1.keys():
       for name,value in w1.items():
          if name == "lat":
               Lat = value
          elif name == "lon":
               Lon = value
elif w1.tag == "type":
   GType = w1.text
elif w1.tag == "link":
   if w1.keys():
      for name, value in w1.items():
          Info = value
   Link = w1.text
```

```
print "Cache Name: ",CacheName
print "Cache ID: ",CacheID
print "Open Cache: ",OpenCache
print "Cache Type: ",CacheType
print "Cache Size: ",CacheSize
print "Difficulty: ", Difficulty
print "Terrain: ",Terrain
print "Lat: ",Lat
print "Lon: ",Lon
print "GType: ",GType
print "Link: ",Link
```

```
import elementtree.ElementTree as ET
tree = ET.parse('Cache.loc')
w = tree.find('.//waypoint')
for w1 in w:
    if w1.tag == "name":
        # Get text of cache name up to the phrase "Open Cache:
"
        CacheName = w1.text[:w1.text.find("Open Cache: ")-1]
        # Get the text between "Open Cache: " and "Cache Type:
"
        OpenCache = w1.text[w1.text.find("Open Cache:
")+12:w1.text.find("Cache Type: ")-1]
        # More of the same
        CacheType = w1.text[w1.text.find("Cache Type:
")+12:w1.text.find("Cache Size: ")-1]
        CacheSize = w1.text[w1.text.find("Cache Size:
")+12:w1.text.find("Difficulty: ")-1]
        Difficulty= w1.text[w1.text.find("Difficulty:
")+12:w1.text.find("Terrain   : ")-1]
        Terrain =   w1.text[w1.text.find("Terrain   : ")+12:]
        if w1.keys():
            for name,value in w1.items():
                if name == 'id':
                    CacheID = value
    elif w1.tag == "coord":
        if w1.keys():
            for name,value in w1.items():
                if name == "lat":
                    Lat = value
                elif name == "lon":
                    Lon = value
    elif w1.tag == "type":
        GType = w1.text
    elif w1.tag == "link":
        if w1.keys():
            for name, value in w1.items():
                Info = value
        Link = w1.text
print "Cache Name: ",CacheName
print "Cache ID: ",CacheID
print "Open Cache: ",OpenCache
print "Cache Type: ",CacheType
print "Cache Size: ",CacheSize
print "Difficulty: ", Difficulty
print "Terrain: ",Terrain
print "Lat: ",Lat
print "Lon: ",Lon
print "GType: ",GType
print "Link: ",Link
print "="*25

print "finished"
```

Programozzunk Pythonban - 11. rész

Írta: Greg Walters

Előző részek:
FCM 27.-36. számokban
az 1.-10. részek

Itt használható:

ubuntu kubuntu xubuntu

Kategóriák:

Eszközök:

Legutóbb azt ígértem, hogy az XML tudásunkat felhasználva időjárási információkat fogunk leszedni egy weboldalról és egy terminálban jelenítjük meg azokat. Nos, ez most elérkezett.

A www.wunderground.com API-ját fogjuk felhasználni. Már hallom a torkotokból felmorajló kérdést: „Az meg mit jelent, hogy API?”. Az API az Application Programming Interface (azaz alkalmazás programozói interfész) rövidítése. Ez egy divatos kifejezés arra, amikor egy másik programhoz kapcsolódunk. Gondolj az importált függvénykönyvtárakra. Némelyikük önálló alkalmazásként is futtatható, de függvénykönyvtárként importálva a legtöbb funkciót saját programunkban is felhasználhatjuk, s így használatba vehetjük másvalaki kódját. Esetünkben időjárási információkat fogunk lekérdezni a wunderground weboldalról, speciálisan formázott URL címek segítségével anélkül, hogy böngészőt használnánk. Egyesek szerint az API olyan, mint egy másik program titkos hátsó bejárata, amit a programozó(k) direkt nekünk készítettek. Így, vagy úgy, ez az alkalmazásnak egy olyan kiegészítése, amivel annak más alkalmazásokban történő felhasználását segíti elő.

Érdekesen hangzik? Nos, olvass tovább, kedves Padawanom.

Izzítsd be a kedvenc böngésződet és irány a www.wunderground.com. Most írd be az irányítószámod, vagy egy várost és egy államot (vagy egy országot) a kereső mezőbe. Itt rengeteg információt találsz. Most ugorjunk az API weboldalára: http://wiki.wunderground.com/index.php/API_-_XML

Az egyik első dolog, amit észreveszel, az API felhasználási feltételei. Kérlek, olvasd el, és tartsd be. Nem fárasztóak és igazán könnyen betarthatók. Számunkra a GeoLookupXML, WXCurrentObXML, AlertsXML és ForecastXML hívások lesznek érdekesek. Szánj egy kis időt az átolvasásukra.

A GeoLookupXML rutin tanulmányozását rád bízom. Két másik parancsra összpontosítunk, most a WXCurrentObXML-re (aktuális állapot) és legközelebb a ForecastXML-re (előrejelzés).

Itt a link a WXCurrentObXML-hez:
http://api.wunderground.com/auto/wui/geo/WXCurrentObXML/index.xml?query=80013

Az Amerikai Egyesült Államokban a 80013-as irányítószámot írd át a sajátodra, vagy ha más országban élsz, megpróbálhatsz várost és országot megadni így: Paris, France, vagy London, England.

Itt a ForecastXML-hez tartozó link:
http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=80013

Itt is írd át a 80013-as irányítószámot a sajátodra, vagy adj meg egy várost és egy országot.

Kezdjük az aktuális információkkal. Másold a címet a kedvenc böngésződbe. Temérdek információt kapsz cserébe. Eldöntheted, melyikeket tartod igazán fontosnak, de mi csak pár elemet fogunk megvizsgálni.

Példánkban a következő címkékre fordítunk figyelmet:

display_location

observation_time
weather
temperature_string
relative_humidity
wind_string
pressure_string

Ezt a listát természetesen bővítheted, ha más címkékre is kíváncsi vagy. Példánk azonban ezekkel a címkékkel is elegendő támpontot nyújt majd tetszőleges irányú és mértékű folytatáshoz.

Most, hogy tudjuk, mit fogunk keresni, kezdjük el lekódolni az alkalmazásunkat. Nézzük a programot nagy vonalakban.

Először megvizsgáljuk, mit kért tőlünk a felhasználó. Ha megadott egy címet, akkor azzal, egyébként pedig a főprogramban rögzített, alapértelmezett címmel dolgozunk. Ezt átadjuk a getCurrents rutinnak. A címet beépítjük a webes lekérdezésbe. A válasz fogadására és objektummá alakítására az urllib.urlopen-t használjuk, majd a létrejött objektumot átadjuk az ElementTree függvénykönyvtár parse függvényének. Ezután lezárjuk a webes kapcsolatot és elkezdjük megkeresni a címkéinket. Találat esetén a címke szövegét elmentjük egy változóba, amiből később majd kiírhatjuk a kimenetre. Amint megvan minden adatunk, megjelenítjük őket. Elég egyszerű koncepció.

Kezdjük azzal, hogy a fájlunknak a w_currents.py nevet adjuk. Itt a kódunk import része:

```
from xml.etree import
ElementTree as ET

import urllib

import sys

import getopt
```

Aztán írjunk egy pár sor súgószöveget (jobbra fent) az importok fölé.

Mindenképpen tripla idézőjeleket használj. Így lehet többsoros kommenteket készíteni. Ezt a részt később még egy kicsit részletezzük.

Most hozzuk létre az osztályok vázát (jobbra lent) és a következő oldalon látható fő rutinokat.

```
""" w_currents.py
Returns current conditions, forecast and alerts for a
given zipcode from WeatherUnderground.com.
Usage: python wonderground.py [options]
Options:
-h, --help Show this help
-l, --location City,State to use
-z, --zip Zipcode to use as location

Examples:
w_currents.py -h (shows this help information)
w_currents.py -z 80013 (uses the zip code 80013 as
location)
"""
```

```
class CurrentInfo:
"""
This routine retrieves the current condition xml data
from WeatherUnderground.com
based off of the zip code or Airport Code...
currently tested only with Zip Code and Airport code
For location,
if zip code use something like 80013 (no quotes)
if airport use something like "KDEN" (use double-quotes)
if city/state (US) use something like "Aurora,%20CO" or
“Aurora,CO” (use double-quotes)
if city/country, use something like "London,%20England"
(use double-quotes)
"""
def getCurrents(self,debuglevel,Location):
pass

def output(self):
pass
def DoIt(self,Location):
pass

#=======================================
# END OF CLASS CurrentInfo()
#=======================================
```

Korábbi cikkekből biztosan emlékszel az „if __name__” sorra. Ha önálló alkalmazásként hívjuk meg ezt a sort, akkor a main rutin fog lefutni - ellenkező esetben programunkat egy függvénykönyvtár részeként használjuk fel. A main rutinba kerülve megvizsgáljuk a kapott paramétereket, ha egyáltalán van ilyen.

Ha a felhasználó a „-h”, vagy „--help” paramétert adta meg, akkor a programkód elején lévő tripla-idézőjeles súgósorokat írjuk ki. Ezt a usage rutin hajtja végre a __doc__ kiíratásával.

Ha a felhasználó a „-l” (cím), vagy a „-z” (irányítószám) kapcsolót adja meg, azzal felülírja a beépített címet. Ha címet adsz meg, mindenképpen tedd idézőjelek közé és soha ne használj szóközöket. Például, a Texas-i Dallasra vonatkozó aktuális értékeket megkaphatod a -l "Dallas,Texas" paraméterezéssel.

Szemfüles olvasóink biztosan észrevették, hogy a -z és -l kapcsolók nagyjából azonosak. Az -l működését átalakíthatod úgy, hogy megkeresve a szóközöket, újraformázza a karakterláncot, mielőtt átadja a rutinnak. Ezt akár most meg is teheted.

Végül létrehozunk egy példányt a CurrentInfo osztályunkból, amit currents-nek nevezünk el, és a címet átadjuk a „DoIt” rutinnak. Töltsük is ki nyomban:

```
def DoIt(self,Location):

self.getCurrents(1,Location)

self.output()
```

Nagyon egyszerű. A címet és a debug levelt átadjuk a getCurrents rutinnak, majd meghívjuk az output rutint. Bár kiírathatnánk az eredményt egyszerűen közvetlenül a getCurrents-ből is, rugalmasabbá tesszük a programunkat azáltal, ha szükség esetén másmás módon is kiírathatjuk az eredményt.

A getCurrents forrása a következő oldalon található.

Van itt egy debuglevel nevű paraméterünk. Ennek segítségével hasznos információkat írathatunk ki, ha a dolgok nem pont úgy alakulnának, ahogy szerettük volna. Ez a programozás korai szakaszában is hasznos lehet. Ha a programod működése már teljes megelégedéssel tölt el, minden debuglevellel kapcsolatos részt törölhetsz. Mielőtt közzéteszed a programot a nagyérdemű számára, mint minden hasonló esetben, ezt a kódot feltétlenül távolítsd el, majd mégegyszer teszteld le, mielőtt útjára indítod.

Most egy try/except wrapperrel fogjuk biztosítani a programunkat lefagyás ellen, ha valami rosszul sülne el. A try oldalon beállítjuk az URL-t és

```
def usage():
print __doc__
def main(argv):
location = 80013
try:
opts, args = getopt.getopt(argv, "hz:l:", ["help=",
"zip=", "location="])
except getopt.GetoptError:
usage()
sys.exit(2)
for opt, arg in opts:
if opt in ("-h", "--help"):
usage()
sys.exit()
elif opt in ("-l", "--location"):
location = arg
elif opt in ("-z", "--zip"):
location = arg
print "Location = %s" % location
currents = CurrentInfo()
currents.DoIt(location)

#============================================
# Main loop
#============================================
if __name__ == "__main__":

main(sys.argv[1:])
```

egy 8 másodperces határidőt (urllib.socket.setdefaulttimeout (8)). Ezt azért tesszük, mert a wunderground néha túlterhelt és nem válaszol. Így nem fogunk ülni és várni a webre a végtelenségig. Ha többet szeretnél megtudni az urllibről, jó kiindulópont a http://docs.python.org/library/urllib.html.

Ha bármi váratlan történik, átkerülünk az except részbe, kiírunk egy hibaüzenetet, és kilépünk (sys.exit(2)).

Feltételezve, hogy minden működik, nekilátunk a címkék keresésének. Elsőként a címünket keressük ezzel az utasítással: tree.findall("//full"). Emlékezzünk csak, a tree az elementtree elemzéséből származó objektum. Hogy pontosan mit ad vissza a website API, azt az alábbiakban láthatjuk.

Ez a <full> címke első előfordulása, ami esetünkben az „Aurora, CO” értéket viseli. Ezt akarjuk címünkként felhasználni. Ezután az „observation_time” címkét keressük, azaz azt az időpontot, amikor az aktuális információk rögzítve lettek. Ugyanezzel a módszerrel keressük meg az összes szükséges információt.

Végül meghívjuk az output rutinunkat, ami a következő oldal bal felső részén található.

Itt egyszerűen kiírjuk a változókat.

Ennyi az egész. Egy példa kimenet az én irányítószámommal és 1-es debuglevel

```
<display_location>
<full>Aurora, CO</full>
<city>Aurora</city>
<state>CO</state>
<state_name>Colorado</state_name>
<country>US</country>
<country_iso3166>US</country_iso3166>
<zip>80013</zip>
<latitude>39.65906525</latitude>
<longitude>-104.78105927</longitude>
<elevation>1706.00000000 ft</elevation>
</display_location>
```

```
def getCurrents(self,debuglevel,Location):
if debuglevel > 0:
print "Location = %s" % Location
try:
CurrentConditions =
'http://api.wunderground.com/auto/wui/geo/WXCurrentObXML
/index.xml?query=%s' % Location
urllib.socket.setdefaulttimeout(8)
usock = urllib.urlopen(CurrentConditions)
tree = ET.parse(usock)
usock.close()
except:
print 'ERROR - Current Conditions - Could not get
information from server...'
if debuglevel > 0:
print Location
sys.exit(2)
# Get Display Location
for loc in tree.findall("//full"):
self.location = loc.text
# Get Observation time
for tim in tree.findall("//observation_time"):
self.obtime = tim.text
# Get Current conditions
for weather in tree.findall("//weather"):
self.we = weather.text
# Get Temp
for TempF in tree.findall("//temperature_string"):
self.tmpB = TempF.text
#Get Humidity
for hum in tree.findall("//relative_humidity"):
self.relhum = hum.text
# Get Wind info
for windstring in tree.findall("//wind_string"):
self.winds = windstring.text
# Get Barometric Pressure
for pressure in tree.findall("//pressure_string"):
self.baroB = pressure.text
```

getCurrents rutin

```
def output(self):
print 'Weather Information From Wunderground.com'
print 'Weather info for %s ' % self.location
print self.obtime
print 'Current Weather - %s' % self.we
print 'Current Temp - %s' % self.tmpB
print 'Barometric Pressure - %s' % self.baroB
print 'Relative Humidity - %s' % self.relhum
print 'Winds %s' % self.winds
```

értékkel az oldal alján látható.

Vedd figyelembe, kérlek, hogy olyan tageket választottam, amelyek a Farenheit és Celsius értékeket is tartalmazzák. Ha például csak a Celsius értékeket szeretnéd megjeleníteni, a <temperature_string> címke helyett a <temp_c> címkét is használhatod.

A teljes forrás letölthető a: http://pastebin.com/4ibJGm74 címről.

Legközelebb az API „előrejelzés” részére koncentrálunk. Addig is, jó szórakozást!

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

```
Location = 80013
Weather Information From Wunderground.com
Weather info for Aurora, Colorado
Last Updated on May 3, 11:55 AM MDT
Current Weather - Partly Cloudy
Current Temp - 57 F (14 C)
Barometric Pressure - 29.92 in (1013 mb)
Relative Humidity - 25%
Winds From the WNW at 10 MPH
Script terminated.
```

Hogyanok

Programozzunk Pythonban - 12. rész

Írta: Greg Walters

Előző részek:
FCM 27.-37. számokban
az 1.-11. részek

Itt használható:

ubuntu kubuntu xubuntu

Kategóriák:

Fejlesztés Grafika Internet M/média Rendszer

Eszközök:

CD/DVD Merevlemez USB eszköz Laptop Vezeték nélküli

Előző alkalommal megnéztük a wunderground API-ját és írtunk is egy kis kódot az aktuális időjárás lekérdezéséhez. Most az API időjárás előrejelző részével fogunk foglalkozni. Ha nem volt alkalmunk elolvasni az előző két, XML-lel foglalkozó cikket - különösképpen az előzőt - akkor, mielőtt továbblépnénk mindenképpen fussuk át őket.

Mint ahogy az aktuális időjárásnak is volt egy webcíme, úgy az előrejelzésnek is van. Az előrejelzés XML oldala: http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=80013

Akárcsak múltkor, most is a „80013”-at meg tudjuk változtatni a mi Város/Országunkra, Város/Államunkra vagy irányítószámunkra. Valószínűleg 600 sornyi XML kódot fogunk visszakapni. A gyökérelem a „forecast” (előrejelzés), melynek négy gyermekeleme van: „termsofservice” (a szolgáltatás feltételei), „txt_forecast” (szöveges előrejelzés), „simpleforecast” (egyszerű előrejelzés) és „moon_phase” (holdfázis). Mi csak a „txt_forecast”-ra és a „simpleforecast”-ra fogunk szorítkozni.

Mivel már átnéztük a usage, main és „if __name__” részeket, ezért ezeket rátok hagyom, és most csak az igazán érdekes részekre fogunk koncentrálni. Már mutattam egy darabkát a txt_forecast-ból, ezért kezdjünk azzal.

Lent látható a környékem txt_forecastjának egy kis szelete.

A txt_forecast szülőelem után találunk egy „date” (dátum), és egy „number” (szám) elemet, majd egy olyan forecastday nevű tagot aminek vannak saját gyermekei, ezek: period (ciklus), icon (ikon), icons (ikonok), title (cím) és valami fcttext nevű dolog, aztán ezek ismétlődnek. Az első dolog amit észrevehetünk, az az, hogy a txt_forecast alatt a dátum valójában egy időpont. Ez gyakorlatilag az előrejelzés kiadásának ideje. A <number> tag mondja meg, hogy mennyi előrejelzés van az elkövetkező 24 órára. Nem emlékszem olyan esetre, amikor ez az érték kettőnél kevesebb lett volna. Minden 24 órás periódushoz (<forecastday>) tartozik egy ciklus szám, többféle ikon beállítás, egy cím opció („Today” [ma], „Tonight” [ma este], „Tomorrow” [holnap]), és a „simple forecast” (rövid előrejelzés) szövege. Ez egy gyors összeg-

```
<txt_forecast>
              <date>3:31 PM MDT</date>
              <number>2</number>
              -<forecastday>
                            <period>1</period>
                            <icon>nt_cloudy</icon>
                            +<icons></icons>
                            <title>Tonight</title>
                            -<fcttext>
                            Mostly cloudy with a 20
percent chance of thunderstorms in the evening...then
partly cloudy after midnight. Lows in the mid 40s.
Southeast winds 10 to 15 mph shifting to the south after
midnight.
                      </fcttext>
              </forecastday>
              +<forecastday></forecastday>
</txt_forecast>
```

zése a következő 12 óra időjárásának.

Mielőtt nekilátnánk kódolni, nem ártana egy pillantást vetni az xml <simpleforecast> részére, mely jobbra látható.

Van egy <forecastday> tagünk minden egyes időjárási ciklushoz, mely általában hat napot jelent, beleértve a mait is. Az alábbi adatok állnak rendelkezésünkre: dátum többféle formátumban (én a <pretty> elemet szeretem), az előrejelzett hőmérsékletek Fahrenheitben és Celsiusban, átlagos körülmények, különböző ikonok, egy égbolt ikon (az elemzőállomás égboltjának állapota), és a „pop”, ami a „Probability Of Percipitation” (csapadékvalószínűség). Néhány érdekes információt szolgáltat a <moon_phase> tag, mint például a naplemente, napfelkelte és hold információkat.

Most nézzük a kódot. Itt van az import szakasz:

```
from xml.etree import
ElementTree as ET

import urllib
import sys

import getopt
```

Most kezdődhet a mai előadás. Létrehozunk egy __init__ szubrutint a számunkra szükséges változók beállításához és kitörléséhez. Ezt jobbra fenn a következő oldalon láthatjuk.

Ha nem érdekel minket mindkét hőmérséklet használata (Fahrenheit és Celsius), akkor hagyjuk ki a megfelelő változócsoportot. Én mindkettőt alkalmazni fogom.

Ezután elkészítjük a lekérdezés metódusát, mely letölti az előrejelzést. Ez látható a következő oldalon lent.

Mindez nagyon hasonló a múltkori, jelenlegi állapotokat lekérdező rutinhoz. Az egyetlen nagy különbség (eddig) a használt URL-ben van. Ezen a ponton azonban néhány dolog megváltozik. Mivel több olyan gyermek is van, melynek a szülőn belül ugyanaz a neve, ezért a parseoló hívásokat egy kissé máshogy kell használni. A kód a következő oldalon, balra fenn látható.

```
<simpleforecast>
    -<forecastday>
        <period>1</period>
        -<date>
            <epoch>1275706825</epoch>
            <pretty_short>9:00 PM MDT</pretty_short>
            <pretty>9:00 PM MDT on June 04, 2010</pretty>
            <day>4</day>
            <month>6</month>
            <year>2010</year>
            <yday>154</yday>
            <hour>21</hour>
            <min>00</min>
            <sec>25</sec>
            <isdst>1</isdst>
            <monthname>June</monthname>
            <weekday_short/>
            <weekday>Friday</weekday>
            <ampm>PM</ampm>
            <tz_short>MDT</tz_short>
            <tz_long>America/Denver</tz_long>
        </date>
        -<high>
            <fahrenheit>92</fahrenheit>
            <celsius>33</celsius>
        </high>
        -<low>
            <fahrenheit>58</fahrenheit>
            <celsius>14</celsius>
        </low>
        <conditions>Partly Cloudy</conditions>
        <icon>partlycloudy</icon>
        +<icons>
        <skyicon>partlycloudy</skyicon>
        <pop>10</pop>
    </forecastday>
            ...
</simpleforecast>
```

```
#================================
# Get the forecast for today and (if available)
tonight
#================================
 fcst = tree.find('.//txt_forecast')
 for f in fcst:
     if f.tag == 'number':
         self.periods = f.text
     elif f.tag == 'date':
         self.date = f.text
     for subelement in f:
         if subelement.tag == 'period':
             self.period=int(subelement.text)
         if subelement.tag == 'fcttext':

self.forecastText.append(subelement.text)
         elif subelement.tag == 'icon':
             self.icon.append( subelement.text)
         elif subelement.tag == 'title':
             self.Title.append(subelement.text)
```

```
class ForecastInfo:
    def __init__(self):
        self.forecastText = []  # Today/tonight forecast
information
        self.Title = []         # Today/tonight
        self.date = ''
        self.icon = []          # Icon to use for conditions
today/tonight
        self.periods = 0
        self.period = 0
       #=============================================
       # Extended forecast information
       #=============================================
        self.extIcon = []       # Icon to use for extended
forecast
        self.extDay = []        # Day text for this forecast
("Monday", "Tuesday" etc)
        self.extHigh = []       # High Temp. (F)
        self.extHighC = []      # High Temp. (C)
        self.extLow = []        # Low Temp. (F)
        self.extLowC = []       # Low Temp. (C)
        self.extConditions = [] # Conditions text
        self.extPeriod = []     # Numerical Period information
(counter)
        self.extpop = []        # Percent chance Of
Precipitation
```

```
 def GetForecastData(self,location):
         try:
             forecastdata = 'http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=%s' % location
             urllib.socket.setdefaulttimeout(8)
             usock = urllib.urlopen(forecastdata)
             tree = ET.parse(usock)
             usock.close()
         except:
             print 'ERROR - Forecast - Could not get information from server...'
             sys.exit(2)
```

Vegyük észre, hogy egy tree.find hívást és egy for ciklust használunk az adatok bejárásához. Szomorú dolog, hogy a Pythonban - más nyelvekkel ellentétben - SELECT/CASE vezérlési szerkezet nincs. Az IF/ELIF megoldás azonban elég jól működik, csak egy kicsit ormótlanabb. Nézzük meg a kódot közelebbről. Az fcst változóhoz hozzárendeljük a <txt_forecast> elemben lévő összes adatot. Ezzel meglesz minden információ az adott csoportból. Ezután megkeressük a <date> és <number> tagokat - mivel ők egyszerű „első szintű” elemek - és betöltjük őket a változóinkba. Ekkor a dolgok egy kicsit tovább bonyolódnak. Vessünk ismét egy pillantást a kapott xml példánkra. A <forecastday> elemnek két példánya van. A <forecastday> alatt lévő alelemek a <period>, <icon>, <icons>, <title> és <fcttext>. Egy ciklussal átiterálunk rajtuk, miközben ismét az IF utasítást használjuk a változókba való betöltésükhöz.

Ezt követően meg kell néznünk az elkövetkező X nap részletesebb előrejelzését. Gyakorlatilag ugyanazt a módszert használjuk. Ezt jobbra láthatjuk.

Létre kell hoznunk az output (kimenet) rutint. Ez - akárcsak múltkor - most is elég általános lesz. A kódot a következő oldalon találjuk.

Ismét, ha nem akarjuk mindkét hőmérséklet - a Celsiust és Fahrenheitet- is használni, akkor módosítsuk a kódot megfelelően. Végül van egy „DoIt” szubrutinunk:

```
def
DoIt(self,Location,US,Include
Today,Output):

      self.GetForecastData(Lo
cation)
     self.output(US,IncludeT
oday,Output)
```

Most a következőképpen hívhatjuk a rutint:

```
forecast = ForecastInfo()

forecast.DoIt('80013',1,0,0)
# Insert your own postal code
```

Ennyi lenne mostanra. Az alert részt meghagyom nektek, ha esetleg végig akarnátok menni rajta.

```
#================================
        # Now get the extended forecast
#================================
    fcst = tree.find('.//simpleforecast')
    for f in fcst:
         for subelement in f:
            if subelement.tag == 'period':
                self.extPeriod.append(subelement.text)
            elif subelement.tag == 'conditions':
                self.extConditions.append(subelement.text)
            elif subelement.tag == 'icon':
                self.extIcon.append(subelement.text)
            elif subelement.tag == 'pop':
                self.extpop.append(subelement.text)
            elif subelement.tag == 'date':
                for child in subelement.getchildren():
                    if child.tag == 'weekday':
                        self.extDay.append(child.text)
            elif subelement.tag == 'high':
                for child in subelement.getchildren():
                    if child.tag == 'fahrenheit':
                        self.extHigh.append(child.text)
                    if child.tag == 'celsius':
                        self.extHighC.append(child.text)
            elif subelement.tag == 'low':
                for child in subelement.getchildren():
                    if child.tag == 'fahrenheit':
                        self.extLow.append(child.text)
                    if child.tag == 'celsius':
                        self.extLowC.append(child.text)
```

A teljes működő kód itt található:
http://pastebin.com/wsSXMXQx

Jó szórakozást a következő alkalomig.

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

```
 def output(self,US,IncludeToday,Output):
     # US takes 0,1 or 2
     #   0 = Centigrade
     #   1 = Fahrenheit
     #   2 = both (if available)
     # Now print it all
     if Output == 0:
         for c in range(int(self.period)):
             if c <> 1:
                 print '------------------------------'
             print 'Forecast for %s' %
self.Title[c].lower()
             print 'Forecast = %s' %
self.forecastText[c]
             print 'ICON=%s' % self.icon[c]
             print '------------------------------'
         print 'Extended Forecast...'
         if IncludeToday == 1:
             startRange = 0
         else:
             startRange = 1
         for c in range(startRange,6):
             print self.extDay[c]
             if US == 0: #Centigrade information
                 print '\tHigh - %s(C)' %
self.extHigh[c]
                 print '\tLow - %s(C)' % self.extLow[c]
             elif US == 1: #Fahrenheit information
                 print '\tHigh - %s(F)' %
self.extHigh[c]
                 print '\tLow - %s(F)' % self.extLow[c]
             else: #US == 2 both(if available)
                 print '\tHigh - %s' % self.extHigh[c]
                 print '\tLow - %s' % self.extLow[c]
             if int(self.extpop[c]) == 0:
                 print '\tConditions - %s.' %
self.extConditions[c]
             else:
                 print '\tConditions - %s.  %d%% chance
of precipitation.' %
(self.extConditions[c],int(self.extpop[c]))
```

Programozzunk Pythonban - 13. rész

Írta: Greg Walters

Ebben a hónapban a Curses Pythonban való használatáról fogunk beszélgetni. Nem, nem arról lesz szó, hogy hogyan káromkodjunk Pythonul (curseing = káromkodás, ford.), de akár erre is vetemedhetünk, ha szükségét éreznénk. Most azonban a Curses modul használatát nézzük meg, mellyel cifrábbá tehetjük a képernyőkimenetet.

Ha elég idősek vagyünk, akkor emlékezhetünk a számítógépek korai időszakára, amikor még üzleti nagygépeink voltak, és buta kis terminálokkal (képernyő és billentyűzet) lehetett be-, illetve kiírni adatokat. Több terminál is csatlakozhatott egy számítógéphez. A gond csak az volt, hogy a számítógép-terminálok nem voltak túl okosak. Nem általak rendelkezésünkre se ablakok, se színek, se semmi, csak 24 darab, 80 karakternyi hosszú sor (legjobb esetben). Még a DOS és a CP/M időszakában - amikor a személyi számítógépek elterjedtek - is csak ennyink volt. Amikor a programozó bonyolultabb kiíratáson dolgozott (legalábbis ahhoz az időszakhoz képest) - kiváltképpen adat ki- és beíráskor - négyzetrácsos papírt használtak a képernyő megtervezéséhez. Minden egyes négyzet egy karakterpozíciónak felelt meg. Amikor terminálban futó Python programmal dolgozunk, még mindig a 24x80-as kijelzővel kell megküzdenünk. Habár, ez az akadály könnyen legyőzhető némi előrelátással és előkészülettel. Tehát, csak el kell mennünk venni egy-két füzetet a helyi papír-írószer boltból.

Mindenesetre vágjunk bele első Curses programunkba, ami jobbra fenn látható. Miután vetettünk egy pillantást a kódra, elmagyarázom az egészet.

```
#!/usr/bin/env python
# CursesExample1
#--------------------------------
# Curses Programming Sample 1
#--------------------------------
import curses
myscreen = curses.initscr()
myscreen.border(0)
myscreen.addstr(12, 25, "See Curses, See Curses Run!")
myscreen.refresh()
myscreen.getch()
curses.endwin()
```

Rövid és egyszerű. Ki is elemezzük soronként. Az elsőben letudjuk az importjainkat, amiknek már elég ismerőseknek kellene lenniük mostanra. Ezután létrehozzuk az új Curses képernyő objektumot, inicializáljuk, majd meghívjuk a myscreen objektumot (myscreen = curses.initscr()). Ez olyan mint egy vászon, amire majd festeni fogunk. Ezt követően használjuk a myscreen.border(0) hívást egy szegély megrajzolásához a képernyő körül. Mindez nem kötelező, de jobban fog tőle kinézni a kijelző. Aztán a Curses addstr() metódusát használjuk egy szöveg „kiírásához” a 12-ik sor 25-ik pozíciójától. Végül meghívjuk a refresh() tagfüggvényt, ami láthatóvá teszi munkánkat. Ha nem frissítenénk a képernyőt, akkor a változtatások nem lennének érzékelhetők. Ezután megvárjuk, amíg a felhasználó megnyom egy gombot (getch), majd felszabadítjuk a kijelző objektumot (endwin), hogy a terminál a szokásos módon tudjon működni. A curses.endwin() hívás egy KIEMELTEN fontos dolog, mivel a konzolunk nagyon ramaty állapotban lesz, ha nincs meghívva. Magyarul, sose feleljtsük el ezt a metódust meghívni mielőtt befejeznénk az alkalmazásunk futtatását.

Mentsük el a programot CursesExample1.py néven és futtassuk egy terminálban. Néhány dolgot érdemes megjegyezni. Mindig, amikor segédlyeket használunk, a kerettel elvesztünk egy „használható”

karakterpozíciót. Továbbá, mind a sor, mind a karakter pozíció száma NULLÁVAL kezdődik. Ez azt jelenti, hogy az első sor a 0.-ik, az utolsó a 23.-ik. Azaz, a felső bal sarok pozíciója 0;0 és az alsó jobbé 23;79. Nézzünk meg egy rövid példát (jobbra fenn), mely ezt mutatja be.

A try/finally blokkok kivételével elég egyszerű dolgok vannak itt. Emlékezzünk, hogy a curses.endwin NAGYON fontos és mindig meg kell hívni kilépés előtt. Ezzel a módszerrel azonban, még ha minden rosszul megy is, az endwin rutin meg fog hívódni. Ugyanez sok más módon megoldható, de számomra ez a legkézenfekvőbb.

Most készítsünk egy jól kinéző menürendszert. Ha visszagondolunk néhány számmal korábra (8. rész), akkor emlékezhetünk a szakácskönyves alkalmazásra, melynek volt egy menüje. Amikor kiírattunk valamit, akkor minden egyszerűen fentebb csúszott. Most ezt az ötletet felhasználva létrehozunk egy „látszatmenüt”, amit később akár fel is használhatunk a szakácskönyves alkalmazás kicsinosításához. Lent látható a régi megoldás.

Ez alkalommal a Cursest fogjuk használni. Kezdjük az alábbi sablonnal. Lehet, hogy hasznos lenne elmenteni ezt a kóddarabkát (jobbra lenn) az esetleges jövőbeli programjainkhoz.

Ahhoz, hogy a fájlon

```
#!/usr/bin/env python
# CursesExample2
import curses
#================================================================
#                          MAIN LOOP
#================================================================
try:
    myscreen = curses.initscr()
    myscreen.clear()
    myscreen.addstr(0,0,"0         1         2         3
4         5         6         7")
    myscreen.addstr(1,0,"12345678901234567890123456789012345
6789012345678901234567890123456789012345678901234567890")
    myscreen.addstr(10,0,"10")
    myscreen.addstr(20,0,"20")
    myscreen.addstr(23,0, "23 - Press Any Key to Continue")
    myscreen.refresh()
    myscreen.getch()
finally:
    curses.endwin()
```

```
#!/usr/bin/env python
#------------------------------
# Curses Programming Template
#------------------------------
import curses

def InitScreen(Border):
    if Border == 1:
        myscreen.border(0)

#================================================================
#                          MAIN LOOP
#================================================================
myscreen = curses.initscr()
InitScreen(1)
try:
    myscreen.refresh()
    # Your Code Stuff Here...
    myscreen.addstr(1,1, "Press Any Key to Continue")
    myscreen.getch()
finally:
    curses.endwin()
```

```
================================================
                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 ->
```

dolgozhassunk a sablon módosítása nélkül, előbb mentsük el „cursesmenu1.py” néven.

Mielőtt továbblépnénk a kóddal, megnézzük az egészet lépésekben. Itt (jobbra fenn) látható a feladatunk pszeudo-kód alakja.

Természetesen ez a pszeudo-kód csak egy álkód. Ennek ellenére elég jó arra, hogy megfelelő kép alakuljon ki bennünk az egész dologról. Mivel mindez csak egy példa, ezért nem is elemezzük ki jobban, de ha szükségét érezzük, akkor elmélyülhetünk benne. Kezdjük a main ciklussal (középen jobbra).

Nem sok programozni való akad itt. Akárcsak a sablonban, itt is megvannak a try|finally blokkjaink. Inicializáljuk a Curses kijelzőt, majd meghívjuk a LogicLoop rutint. Ez a kód jobbra lenn látható.

Mivel ez is csak egy példa, ismét nem sok látnivaló akad. Két rutint hívunk meg. Az egyiket DoMainMenunek, a másikat MainInKey-nek nevezik. (Jobbra lenn látható) A Do-MainMenu

```
curses.initscreen
LogicLoop
    ShowMainMenu                    # Show the main menu
    MainInKey                       # This is our main input handling routine
        While Key != 0:
            If Key == 1:
                ShowAllRecipesMenu  # Show the All Recipes Menu
                Inkey1              # Do the input routines for this
                ShowMainMenu        # Show the main menu
            If Key == 2:
                SearchForARecipeMenu # Show the Search for a Recipe Menu
                InKey2              # Do the input routines for this option
                ShowMainMenu        # Show the main menu again
            If Key == 3:
                ShowARecipeMenu     # Show the Show a recipe menu routine
                InKey3              # Do the input routine for this routine
                ShowMainMenu        # Show the main menu again
          …                     # And so on and so on
curses.endwin()                 # Restore the terminal
```

```
def DoMainMenu():
    myscreen.erase()
    myscreen.addstr(1,1,
"========================================")
    myscreen.addstr(2,1,  "          Recipe
Database")
    myscreen.addstr(3,1,
"========================================")
    myscreen.addstr(4,1,  "  1 - Show All
Recipes")
    myscreen.addstr(5,1,  "  2 - Search for a
recipe")
    myscreen.addstr(6,1,  "  3 - Show a recipe")
    myscreen.addstr(7,1,  "  4 - Delete a recipe")
    myscreen.addstr(8,1,  "  5 - Add a recipe")
    myscreen.addstr(9,1,  "  6 - Print a recipe")
    myscreen.addstr(10,1, "  0 - Exit")
    myscreen.addstr(11,1,
"========================================")
    myscreen.addstr(12,1, "  Enter a selection: ")
    myscreen.refresh()
```

```
#    MAIN LOOP
try:
    myscreen = curses.initscr()
    LogicLoop()
finally:
    curses.endwin()
```

```
def LogicLoop():
    DoMainMenu()
    MainInKey()
```

megjeleníti a főmenüt, a MainInKey pedig lekezel minden mást.

Figyeljük meg, hogy itt képernyőtörlésen (myscreen.erase) és a kiírandó dolgokon kívül más nincs. Sehol nem látunk billentyűzetet kezelő kódot. Ez a MainInKey feladata, ami itt lenn látható.

Ez is egy igen egyszerű rutin. Mindaddig, amíg a beolvasott érték nem 0, egy while ciklusban vagyunk. A ciklusban leellenőrizzük, hogy a kapott adat megegyezik-e különböző értékekkel, és ha igen, akkor egy sor rutint hívunk meg. Végül mikor készen vagyunk, meghívjuk a főmenüt. A legtöbbjüket mostanra már egyedül is meg tudjuk oldani, de a 2-es opciót - Search for a Recipe (Recept keresése) - külön is megnézzük.

```
def MainInKey():
    key = 'X'
    while key != ord('0'):
        key = myscreen.getch(12,22)
        myscreen.addch(12,22,key)
        if key == ord('1'):
            ShowAllRecipesMenu()
            DoMainMenu()
        elif key == ord('2'):
            SearchForARecipeMenu()
            InKey2()
            DoMainMenu()
        elif key == ord('3'):
            ShowARecipeMenu()
            DoMainMenu()
        elif key == ord('4'):
            NotReady("'Delete A Recipe'")
            DoMainMenu()
        elif key == ord('5'):
            NotReady("'Add A Recipe'")
            DoMainMenu()
        elif key == ord('6'):
            NotReady("'Print A Recipe'")
            DoMainMenu()
        myscreen.refresh()
```

```
def SearchForARecipeMenu():
    myscreen.addstr(4,1, "------------------------------")
    myscreen.addstr(5,1, " Search in")
    myscreen.addstr(6,1, "------------------------------")
    myscreen.addstr(7,1, " 1 - Recipe Name")
    myscreen.addstr(8,1, " 2 - Recipe Source")
    myscreen.addstr(9,1, " 3 - Ingredients")
    myscreen.addstr(10,1," 0 - Exit")
    myscreen.addstr(11,1,"Enter Search Type -> ")
    myscreen.refresh()

def InKey2():
    key = 'X'
    doloop = 1
    while doloop == 1:
        key = myscreen.getch(11,22)
        myscreen.addch(11,22,key)
        tmpstr = "Enter text to search in "
        if key == ord('1'):
            sstr = "'Recipe Name' for -> "
            tmpstr = tmpstr + sstr
            retstring = GetSearchLine(13,1,tmpstr)
            break
        elif key == ord('2'):
            sstr = "'Recipe Source' for -> "
            tmpstr = tmpstr + sstr
            retstring = GetSearchLine(13,1,tmpstr)
            break
        elif key == ord('3'):
            sstr = "'Ingredients' for -> "
            tmpstr = tmpstr + sstr
            retstring = GetSearchLine(13,1,tmpstr)
            break
        else:
            retstring = ""
            break
    if retstring != "":
        myscreen.addstr(15,1,"You entered - " + retstring)
    else:
        myscreen.addstr(15,1,"You entered a blank string")
    myscreen.refresh()
    myscreen.addstr(20,1,"Press a key")
    myscreen.getch()

def GetSearchLine(row,col,strng):
    myscreen.addstr(row,col,strng)
    myscreen.refresh()
    instring = myscreen.getstr(row,len(strng)+1)
    myscreen.addstr(row,len(strng)+1,instring)
    myscreen.refresh()
    return instring
```

Ez a menü rövid és egyszerű. Az InKey2 rutin (előző oldalon jobbra) egy kissé bonyolultabb.

Megint a szokásos while ciklust használjuk. A doloop változót 1-re állítjuk, hogy a ciklus addig fusson, amíg nincs meg, amit akartunk. A break hívást használjuk a ciklusból való kilépéshez. A három opció nagyon hasonló. A fő különbség, hogy egy tmpstr nevű változóval kezdünk, majd hozzáfűzzük azt a szöveget, ami ki lett választva, ezzel egy picit barátságosabbá téve azt. Ezután a keresési szöveg bekéréséhez a GetSearchLine-t hívjuk meg. A getstr rutinnal karakterek helyett egy sztringet olvasunk be a felhasználótól. Végül visszaadjuk a karakterláncot az input rutinunknak további feldolgozásra.

A teljes kód itt érhető el: http://pastebin.com/ELuZ3T4P

Még egy utolsó dolog. Ha mélyebben érdekel a Curses programozás, akkor sok más módszer is megtalálható a mostanin felül. Egy Google keresésen kívül a legjobb kiindulópont a hivatalos dokumentáció honlapja lehet, mely a http://docs.python.org/library/curses.html címen érhető el.

Találkozunk legközelebb.

HOPSZ!

Úgy tűnik, hogy a 11. rész kódja a Pastebinen nem volt rendesen indentálva. A helyes kód URL-je ez: http://pastebin.com/Pk74fLF3

Kérlek nézd meg a http://fullcirclemagazine.pastebin.com lapot az összes eddigi (és jövendő) Python kódért.

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

Hogyanok

Programozzunk Pythonban - 14. rész

Írta: Greg Walters

Legutóbb a Curses-ről kezdtünk el beszélni. Most már mélyebbre fogjuk ásni magunkat a témában, kifejezetten a színekkel kapcsolatos parancsokra koncentrálva. Gyorsan összefoglalom a lényeget, arra az esetre, ha nem olvastátok volna az előző cikket. Kezdésként be kell importálni a curses csomagot. Ezután meg kell hívni a curses.initscr metódust. Ahhoz, hogy szöveget írhassunk a képernyőre, a addstr függvényt kell használni, majd a refresh-t a változtatások megjelenítéséhez. Végül, meg kell hívni a curses.endwin()-t a terminál ablak normális állapotra való visszaállításához.

Most egy rövid és egyszerű programot fogunk készíteni, mely színes lesz. Az egész nagyon hasonló az eddigiekhez, annyi csak a különbség, hogy van néhány új parancs. Először a curses.start_color() segítségével tudatjuk a rendszerrel, hogy mely színekre van szükségünk. Ezt követően beállítjuk az előtér és háttér színpárt. Sok fajta párt beállíthatunk, melyeket bármikor fel tudunk használni. Ezt a curses.init_pair függvénnyel tesszük meg. A szintaxis:

```
curses.init_pair([pairnumber]
,[foreground
color],[background color])
```

A „curses.COLOR_” szöveg és a szín megfelelő angol nevének az összefűzésével állíthatjuk be a színeket. Például: curses.COLOR_BLUE vagy curses.COLOR_GREEN. A választási lehetőségeink a következők: fekete (black), piros (red), green (zöld), sárga (yellow), kék (blue), magenta (bíbor), cyan (ciánkék) és white (fehér). Egyszerűen, nagybetűkkel írva fűzzük hozzá a megfelelőt a “curses.COLOR_”-hoz és megkapjuk a színt. Amint beállítottuk a színpárt, felhasználhatjuk a screen.addstr függvény utolsó paramétereként:

```
myscreen.addstr([row],[column
],[text],curses.color_pair(X)
)
```

```
import curses
try:
    myscreen = curses.initscr()
    curses.start_color()
    curses.init_pair(1, curses.COLOR_BLACK,
curses.COLOR_GREEN)
    curses.init_pair(2, curses.COLOR_BLUE,
curses.COLOR_WHITE)
    curses.init_pair(3,
curses.COLOR_MAGENTA,curses.COLOR_BLACK)
    myscreen.clear()
    myscreen.addstr(3,1,"  This is a test
",curses.color_pair(1))
    myscreen.addstr(4,1,"  This is a test
",curses.color_pair(2))
    myscreen.addstr(5,1,"  This is a test
",curses.color_pair(3))
    myscreen.refresh()
    myscreen.getch()
finally:
    curses.endwin()
```

Itt láthatjuk az általunk választott X színhalmazt.

Mentsük el a kódot (fent látható) colortest1.py néven, majd futtassuk. Ne próbálkozzunk curses programok olyan IDE-n belüli futtatásával, mint az SPE vagy a Dr. Python. A terminált használjuk erre.

Egy szürke hátteret kellene kapnunk három sornyi, különböző színű „This is a test” felirattal. Az elsőnek zöldnek feketén, a másodiknak fehéren kéknek, a harmadiknak bíbornak szürke háttéren kellene lennie.

Emlékezzünk még a Try/Finally szerkezetre. Ezzel biztosíthatjuk be magunkat arra az esetre, hogy ha valami nem megfelelő történik és a terminált vissza kell állítani normál állapotba. Van még egy másik

módszer is. A Cursesben található egy wrapper nevű parancs. A wrapper mindent elvégez helyetted. Magától meghívja a curses.initscr()-t, a curses.start_color()-t és a curses.endwin()-t. Egyetlen dolgot kell csak szem előtt tartani: a curses.wrappert a main függvényben kell meghívni. Ez visszaadja neked a screen mutatóját. Jobbra fenn ugyanaz a program látható, csak most a curses.wrapper használatával.

Ezzel a módszerrel nem csak egyszerűbb lesz a program írása, de az endwin meg nem hívódása miatt sem kell aggódni, ha valami hiba merülne fel. Gyakorlatilag minden munkát elvégez helyettünk.

Most, hogy már egy csomó alapvető dolgot megtanultunk, használjuk is fel ezeket, az elmúlt egy év tapasztalatával, egy játék létrehozásához. Mielőtt belevágnánk, beszéljük meg, hogy ez mit is takar. A játék véletlenszerűen választ egy nagybetűt és miközben a képernyő jobb széléről a bal szélére mozgatja, egy véletlen pozíción le fog esni a képernyő alján. A játékosnak lesz egy „fegyvere”, melyet a jobb és bal nyílbillentyűkkel lehet a lehulló betű alá mozgatni. A szóköz megnyomásával lőhetünk. Ha sikerül lelőni a betűt, mielőtt az elérné a fegyverünket, akkor kapunk egy pontot. Ha nem sikerül, akkor a fegyver felrobban. Ha elvesztünk hármat, akkor vége a játéknak. Egyszerű játék, de sok kódot kell megírni hozzá.

Vágjunk is bele. Végezzük el a kezdeti beállításokat, és hozzunk létre egy pár rutint. Kezdjünk egy új projektet, melyet game1.py-nek fogunk hívni. Először nézzük meg a jobbra lent látható kódot.

Ez a kód még nem sokat csinál, de ettől függetlenül, egy jó kiindulópontot jelent számunkra. Figyeljük meg, hogy négy darab init_pair színbeállító utasításunk van, melyeket a véletlenül meghatározott színekhez és a robbanáshoz használunk (ötödik). Most állítsunk be egyket változót és konstansot. Ezeket az Game1 osztály __init__ rutinjában fogjuk elhelyezni. Cseréljük le a pass utasítást a következő oldalon lévő kódra.

```
import curses
def main(stdscreen):
    curses.init_pair(1, curses.COLOR_BLACK,
curses.COLOR_GREEN)
    curses.init_pair(2, curses.COLOR_BLUE,
curses.COLOR_WHITE)
    curses.init_pair(3,
curses.COLOR_MAGENTA,curses.COLOR_BLACK)
    stdscreen.clear()
    stdscreen.addstr(3,1,"  This is a test
",curses.color_pair(1))
    stdscreen.addstr(4,1,"  This is a test
",curses.color_pair(2))
    stdscreen.addstr(5,1,"  This is a test
",curses.color_pair(3))
    stdscreen.refresh()
    stdscreen.getch()
curses.wrapper(main)
```

```
import curses
import random

class Game1():
    def __init__(self):
        pass
    def main(self,stdscr):
        curses.init_pair(1, curses.COLOR_BLACK,
curses.COLOR_GREEN)
        curses.init_pair(2, curses.COLOR_BLUE,
curses.COLOR_BLACK)
        curses.init_pair(3, curses.COLOR_YELLOW,
curses.COLOR_BLUE)
        curses.init_pair(4, curses.COLOR_GREEN,
curses.COLOR_BLUE)
        curses.init_pair(5, curses.COLOR_BLACK,
curses.COLOR_RED)

    def StartUp(self):
        curses.wrapper(self.main)
g = Game1()
g.StartUp()
```

Mostanra már magadtól is ki kellene tudnod találni, hogy ezekben a definíciókban mi történik. Ha nem vagyunk biztosak magunkban, akkor megígérem, hogy miközben kitöltjük, minden megvilágosodik számunkra.

Lassan kapunk egy működő kódot. Ennek ellenére még mindig létre kell hoznunk egy pár metódust, mielőtt valamire használhatnánk is. Nézzük meg a betűt jobbról balra mozgató rutint: http://fullcirclemagazine.pastebin.com/z5CgMAgm

Ez lesz a leghosszabb az egész programban, és találkozhatunk egy-két új függvénnyel is. A scr.delch() metódussal töröljük a karaktert az adott sor|oszlopban. A curses.napms() megmondja a pythonnak, hogy várjon X ezredmásodpercet (ms).

A rutin logikáját (pszeudokódban) a következő oldalon (jobbra fenn) figyelhetjük meg.

Most már képesnek kell lenned a kód végigkövetésére. Két függvényre van csak szükségünk a helyes működéshez. Az első az Explode, amit egyelőre a pass utasítással töltünk ki, a

```
        # Line Specific Stuff
        self.GunLine = 22                #Row where our gun lives
        self.GunPosition = 39            #Where the gun starts on GunLine
        self.LetterLine = 2              #Where our letter runs right to left
        self.ScoreLine = 1               #Where we are going to display the score
        self.ScorePosition = 50          #Where the score column is
        self.LivesPosition = 65          #Where the lives column is

        # Letter Specific Stuff
        self.CurrentLetter = "A"         #A dummy Holder Variable
        self.CurrentLetterPosition = 78 #Where the letter will start on the LetterLine
        self.DropPosition = 10           #A dummy Holder Variable
        self.DroppingLetter = 0          #Flag - Is the letter dropping?
        self.CurrentLetterLine = 3       #A dummy Holder Variable
        self.LetterWaitCount = 15        #How many times should we loop before actually
                                          working?

        # Bullet Specific Stuff
        self.Shooting = 0                #Flag - Is the gun shooting?
        self.BulletRow = self.GunLine - 1
        self.BulletColumn = self.GunPosition

        # Other Stuff
        self.LoopCount = 0               #How many loops have we done in MoveLetter
        self.GameScore = 0               #Current Game Score
        self.Lives = 3                   #Default number of lives
        self.CurrentColor = 1            #A dummy Holder Variable
        self.DecScoreOnMiss = 0          #Set to 1 if you want to decrement the
                                         #score every time the letter hits the
                                         #bottom row
```

második a ResetForNew. Ezzel az aktuális sort, illetve oszlopot állítjuk vissza alapértelmezettre, beállítjuk a DropLetter kapcsolót 0-ra, választunk egy véletlen betűt és megjelenési pontot. A következő oldalon középen jobbra láthatóak.

Még további négy függvényre van szükségünk (következő oldal, jobbra lenn). Az első kiválaszt egy véletlen betűt, a második pedig egy véletlen megjelenési pontot. Emlékszünk még, hogy volt a cikksorozat elején a random modulról szó?

A PickALetterben 65 és 90 között generálunk egy számot (A-tól Z-ig). Emlékezzünk arra, hogy véletlen egész generálásakor meg kell adnunk egy minimum és maximum számpárt. Ugyanez áll a PickDropPoint esetében is. A random.seed()-et is meghívjuk mindkettőben, ami minden alkalommal inicializálja a véletlenszám generátort. A negyedik rutint

CheckKeys-nek hívják. Ebben bármilyen, a felhasználó által megnyomott billentyűt figyelünk, és megfelelően lekezeljük a fegyver mozgatását. Ezt egyelőre megíratlanul hagyjuk. Szükségünk lesz még egy CheckForHit nevű metódusra, melyet ismét csak pass-szel töltünk ki.

```
def
CheckKeys(self,scrn,keyin):
    pass
def CheckForHit(self,scrn):
    pass
```

Létre fogunk hozni egy aprócska rutint, ez lesz a játék „agya”. Ezt GameLoopnak fogjuk hívni (következő oldalon, jobbra fenn).

Itt a mögöttes logika az, hogy először a billentyűzetet nodelay(1)-re állítjuk. Ez azt jelenti, hogy nem várunk billentyű kombinációkra, és ha mégis kapunk egyet, akkor egyszerűen eltároljuk későbbi feldolgozás céljából. Ezt követően belépünk egy while ciklusba, mely mindig igaz (1), így a játék addig fog tartani, amíg úgy nem döntünk, hogy végeztünk. 40 ezredmásodpercig várunk, majd elmozdítjuk a betűt és ellenőrizzük, hogy a felhasználó megnyomott-e egy gombot. Ha ez egy „Q” (nagybetűs), vagy az ESC gomb, akkor megszakítjuk a ciklust és kilépünk a programból. Egyébként leellenőrizzük, hogy a megnyomott gomb jobb, vagy bal nyíl, esetleg szóköz-e. Később úgy nehezíthetjük a játékot, hogy csak akkor lőhetünk, ha az aktuális karakternek megfelelő billentyűt nyomtuk le (akárcsak egy egyszerű, gépelést tanító programban). Csak nehogy elfeledjük a Q-t kivenni (a kilépés gombot).

```
IF we have waited the correct number of loops THEN
   Reset the loop counter
   IF we are moving to the left of the screen THEN
     Delete the character at the the current row,column.
     Sleep for 50 milliseconds
     IF the current column is greater than 2 THEN
          Decrement the current column
     Set the character at the current row,column
     IF the current column is at the random column to drop to the bottom THEN
          Set the DroppingLetter flag to 1
   ELSE
     Delete the character at the current row,column
     Sleep for 50 milliseconds
     IF the current row is less than the line the gun is on THEN
          Increment the current row
          Set the character at the current row,column
     ELSE
          IF
          Explode (which includes decrementing the score if you wish) and check to
           see if we continue.
          Pick a new letter and position and start everything over again.
ELSE
   Increment the loopcounter
Refresh the screen.
```

```
    def Explode(self,scrn):
        pass
    def ResetForNew(self):
        self.CurrentLetterLine = self.LetterLine
        self.CurrentLetterPosition = 78
        self.DroppingLetter = 0
        self.PickALetter()
        self.PickDropPoint()
```

```
    def PickALetter(self):
        random.seed()
        char = random.randint(65,90)
        self.CurrentLetter = chr(char)

    def PickDropPoint(self):
        random.seed()
        self.DropPosition = random.randint(3,78)
```

Egy olyan függvényt is létre kell hoznunk, ami az új játékokat előkészíti. Legyen ennek neve NewGame (jobbra középen).

Szükségünk van még egy rutinra (PrintScore), ami kiíratja az aktuális pontszámot és a maradék életet (jobbra lenn).

Most már csak egy pici kód maradt hátra (balra lenn) a main függvényben, mely elindítja a játék ciklust. További kód lenn látható. Helyezzük el ezt az utolsó init_pair hívás után.

Végre van egy olyan programunk, ami csinál is valamit. Próbáld ki, megvárlak.

Itt a játék már tud választani egy nagybetűt, azt jobbról balra, majd lefelé mozgatni a képernyőn. Ennek ellenére észrevehetjük, hogy bármennyiszer is futtatjuk a programot, az első betű mindig „A” és a megjelenés helye mindig a 10. oszlop. Ez azért van, mert az __init__ rutinban megadtunk alapértelmezett értékeket. Ezt úgy tudjuk kiküszöbölni, hogy egyszerűen meghívjuk a self.ResetForNew metódust, mielőtt belépnénk a main while ciklusába.

Ezen a ponton még dolgoznunk kell egy kicsit a „fegyverünkön” és a többi segédfüggvényen. Helyezzük el a kódot

```
        stdscr.addstr(11,28,"Welcome to Letter Attack")
        stdscr.addstr(13,28,"Press a key to begin....")
        stdscr.getch()
        stdscr.clear()
        PlayLoop = 1
        while PlayLoop == 1:
            self.NewGame(stdscr)
            self.GameLoop(stdscr)
            stdscr.nodelay(0)
            curses.flushinp()
            stdscr.addstr(12,35,"Game Over")
            stdscr.addstr(14,23,"Do you want to play
again? (Y/N)")
            keyin = stdscr.getch(14,56)
            if keyin == ord("N") or keyin == ord("n"):
                break
            else:
                stdscr.clear()
```

```
    def GameLoop(self,scrn):
        test = 1              #Set the loop
        while test == 1:
            curses.napms(20)
            self.MoveLetter(scrn)
            keyin =
scrn.getch(self.ScoreLine,self.ScorePosition)
            if keyin == ord('Q') or keyin == 27:  # 'Q'
or <Esc>
                break
            else:
                self.CheckKeys(scrn,keyin)
            self.PrintScore(scrn)
            if self.Lives == 0:
                break
        curses.flushinp()
        scrn.clear()
```

```
    def NewGame(self,scrn):
        self.GunChar = curses.ACS_SSBS

scrn.addch(self.GunLine,self.GunPosition,self.GunChar,cu
rses.color_pair(2) | curses.A_BOLD)
        scrn.nodelay(1)    #Don't wait for a
keystroke...just cache it.
        self.ResetForNew()
        self.GameScore = 0
        self.Lives = 3
        self.PrintScore(scrn)
        scrn.move(self.ScoreLine,self.ScorePosition)
```

```
    def PrintScore(self,scrn):

scrn.addstr(self.ScoreLine,self.ScorePosition,"SCORE:
%d" % self.GameScore)

scrn.addstr(self.ScoreLine,self.LivesPosition,"LIVES:
%d" % self.Lives)
```

(jobbra fenn) a Game1 osztályban.

A Movegun lekérdezi a fegyver aktuális pozícióját és elmozdítja a megadott irányba. Az egyetlen dolog, ami új, az a végén lévő addch rutin. Meghívjuk a colorpair(2)-t a szín beállításához, majd ugyanitt utasítjuk a fegyvert, hogy félkövér legyen. Ehhez a bitenkénti VAGY-ot („|”) használjuk. Ezután meg kell írnunk a CheckKeys rutint. Cseréljük le a pass utasítást a következő oldalon, jobbra lent lévő kódra.

Még kell egy rutin, ami a lövedéket „felfelé” mozgatja a képernyőn (balra lenn).

Szükségünk van még pár függvényre (következő oldal, jobbra fenn), mielőtt befejezettnek minősíthetnénk a programot. Itt található a CheckForHit és az ExplodeBullet kódja.

Végül, megírjuk az Explode rutint. Helyettesítsük a pass-t a következő kóddal (következő oldal, lenn).

Ez már a kész program. A betűk sebességének lassításához/gyorsításához és a nehézségi szint változtatásához a LetterWaitCount-tal babrálhatunk egy kicsit. Használhatjuk még a CurrentColort véletlen színválasztáshoz és a betű színének a négy színpár valamelyikéhez való véletlen hozzárendeléséhez.

```
    def MoveGun(self,scrn,direction):
        scrn.addch(self.GunLine,self.GunPosition," ")
        if direction == 0:      # left
            if self.GunPosition > 0:
                self.GunPosition -= 1
        elif direction == 1:  # right
            if self.GunPosition < 79:
                self.GunPosition += 1

scrn.addch(self.GunLine,self.GunPosition,self.GunChar,cur
ses.color_pair(2) | curses.A_BOLD)
```

```
            if keyin == 260: # left arrow -  NOT on keypad
                self.MoveGun(scrn,0)
                curses.flushinp()  #Flush out the input buffer for safety.
            elif keyin == 261: # right arrow - NOT on keypad
                self.MoveGun(scrn,1)
                curses.flushinp()  #Flush out the input buffer for safety.
            elif keyin == 52:  # left arrow  ON keypad
                self.MoveGun(scrn,0)
                curses.flushinp()  #Flush out the input buffer for safety.
            elif keyin == 54:  # right arrow ON keypad
                self.MoveGun(scrn,1)
                curses.flushinp()  #Flush out the input buffer for safety.
            elif keyin == 32:  #space
                if self.Shooting == 0:
                    self.Shooting = 1
                    self.BulletColumn = self.GunPosition
                    scrn.addch(self.BulletRow,self.BulletColumn,"|")
                    curses.flushinp()  #Flush out the input buffer for safety.
```

Gondolj úgy erre, mint egy kis házi feladatra.

Remélem élvezted az e havi cikket és ki fogod egy kicsit bővíteni a programot, hogy még játszhatóbb legyen.

```
    def MoveBullet(self,scrn):
        scrn.addch(self.BulletRow,self.BulletColumn," ")
        if self.BulletRow > self.LetterLine:
            self.CheckForHit(scrn)
            self.BulletRow -= 1

scrn.addch(self.BulletRow,self.BulletColumn,"|")
        else:
            self.CheckForHit(scrn)

scrn.addch(self.BulletRow,self.BulletColumn," ")
            self.BulletRow = self.GunLine - 1
            self.Shooting = 0
```

Mint mindig, a teljes kód a www.thedesignatedgeek.com, vagy a: http://fullcirclemagazine.pastebin.com/DeReeh8m címen érhető el.

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

```
    def CheckForHit(self,scrn):
        if self.Shooting == 1:
            if self.BulletRow == self.CurrentLetterLine:
                if self.BulletColumn == self.CurrentLetterPosition:
                    scrn.addch(self.BulletRow,self.BulletColumn," ")

                    self.ExplodeBullet(scrn)
                    self.GameScore +=1
                    self.ResetForNew()

    def ExplodeBullet(self,scrn):
        scrn.addch(self.BulletRow,self.BulletColumn,"X",curses.color_pair(5))
        scrn.refresh()
        curses.napms(200)
        scrn.addch(self.BulletRow,self.BulletColumn,"|",curses.color_pair(5))
        scrn.refresh()
        curses.napms(200)
        scrn.addch(self.BulletRow,self.BulletColumn,"-",curses.color_pair(5))
        scrn.refresh()
        curses.napms(200)
        scrn.addch(self.BulletRow,self.BulletColumn,".",curses.color_pair(5))
        scrn.refresh()
        curses.napms(200)
        scrn.addch(self.BulletRow,self.BulletColumn," ",curses.color_pair(5))
        scrn.refresh()
        curses.napms(200)
```

```
        scrn.addch(self.CurrentLetterLine,self.CurrentLetterPosition,"X",curses.color_pair(5))
        curses.napms(100)
        scrn.refresh()
        scrn.addch(self.CurrentLetterLine,self.CurrentLetterPosition,"|",curses.color_pair(5))
        curses.napms(100)
        scrn.refresh()
        scrn.addch(self.CurrentLetterLine,self.CurrentLetterPosition,"-",curses.color_pair(5))
        curses.napms(100)
        scrn.refresh()
        scrn.addch(self.CurrentLetterLine,self.CurrentLetterPosition,".",curses.color_pair(5))
        curses.napms(100)
        scrn.refresh()
        scrn.addch(self.CurrentLetterLine,self.CurrentLetterPosition," ")
        scrn.addch(self.GunLine,self.GunPosition,self.GunChar,curses.color_pair(2) | curses.A_BOLD)
        scrn.refresh()
```

Hogyanok

Programozzunk Pythonban - 15. rész

Írta: Greg Walters

Ebben a hónapban elkezdjük a Pygame modul felfedezését, melyet kifejezetten játékok írására találtak ki. A weblapja: http://www.pygame.org/. Kapásból idéznék is a Pygame readme-jéből: „A Pygame egy platformfüggetlen program modul, ami olyan multimédiás szoftverek Pythonban való egyszerűsített megírására szolgál, mint például a videojátékok is. A Pygame-hez szükségünk van a Python nyelvre, illetve további közismert programozási könyvtárakra.”

A Pygame-et a Synapticon keresztül a „python-pygame” néven telepíthetjük. Ezt tegyük is meg.

Először beimportáljuk a Pygame-et (lásd jobbra fenn), majd beállítjuk az os.environ változót az ablakunk középre pozícionálásához. Ezt követően inicializáljuk a Pygame ablakot 800x600 pixeles felbontásra, illetve beállítjuk a címsort. Végül megjelenítünk mindent és belépünk egy olyan ciklusba, ami egy billentyű, vagy az egér megnyomására vár. A screen egy olyan objektum, ami minden általunk felhasznált elemet tárol. Ezt felületnek (surface) nevezzük. Gondoljunk erre úgy, mint egy darab papírra, amelyre rajzolni akarunk.

Idáig nem túl izgalmas, de kezdetnek ez is megteszi. Dobjuk fel egy kicsit. A háttérszínt állítsuk valamilyen kevésbé sötét színre. Találtam egy „colorname” nevezetű programot, amit az Ubuntu Szoftverközponton keresztül fel tudunk telepíteni. Segítségével kiválaszthatunk egy színt egy „színkeréken”, és meg fogja adni ennek RGB - azaz piros, zöld és kék – értékét. Ha nem szeretnénk az előredefiniált Pygame-es színekkel dolgozni, akkor mindenképpen RGB színeket kell használnunk. Ez egy ügyes kis segédprogram, mely hasznunkra válhat.

Közvetlenül az import utasítások alá helyezzük el a

```
#This is the Import
import pygame
from pygame.locals import *
import os
# This will make our game window centered in the screen
os.environ['SDL_VIDEO_CENTERED'] = '1'
# Initialize pygame
pygame.init()
#setup the screen
screen = pygame.display.set_mode((800, 600))
# Set the caption (title bar of the window)
pygame.display.set_caption('Pygame Test #1')
# display the screen and wait for an event
doloop = 1
while doloop:
    if pygame.event.wait().type in (KEYDOWN,
MOUSEBUTTONDOWN):
        break
```

```
Background = 208, 202, 104
```

sort. Ezzel beállítjuk a Backgroundot egy cser színre. Következőnek a pygame.display .set_caption sor után írjuk be az alábbiakat:

```
screen.fill(Background)
pygame.display.update()
```

A screen.fill() metódus a megadott értékre állítja a háttérszínt. A következő sor, a pygame.display.update() pedig a képernyőn aktualizálja a változtatásokat.

Mentsük el a programot *pygame1.py* néven és lépjünk tovább.

Most a kellemes kinézetű ablakunkban meg fogunk jeleníteni egy szöveget. Ismét az import utasításainkkal és a háttérváltozók beállításával kezdünk:

```
import pygame
from pygame.locals import *
import os
Background = 208, 202, 104
```

Írjuk meg a betűtípus előtér

színét is:

```
FontForeground = 255,255,255
# White
```

Ezután az előző kódot nagyrészt átemeljük (jobbra).

Ha most futtatjuk, akkor a külalakon semmi változást nem fogunk tapasztalni, mivel kizárólag az előtér színét változtattuk meg. Írjuk be az alábbi kódot a screen.fill() sor és a ciklus közé:

```
font =
pygame.font.Font(None,27)
text = font.render('Here is
some text', True,
FontForeground, Background)
textrect = text.get_rect()
screen.blit(text,textrect)
pygame.display.update()
```

Mentsünk pygame2.py néven és futtassuk. Az ablakunk bal felső sarkában láthatjuk a „Here is some text” feliratot.

Egyenként nézzük meg az új parancsokat. Először a Font metódust hívjuk, melynek két argumentumot adunk át. Az első a betűkészlet neve, a második a betűméret. Egyelőre a 'None' kulcsszó használatával a típusválasztást rábízzuk a rendszerre, a méretet pedig 27 pontra állítjuk.

A következő a font.render() metódus. Ennek négy darab argumentuma van, melyek rendre: a megjelenítendő szöveg, akarunk-e élsimítást használni (ebben az esetben True, azaz igen), végül a betű előtér és háttér színe.

A következő sor (text.get_rect()) lekér egy befoglaló téglalap objektumot, amivel majd kirakjuk a szöveget a képernyőre. Ez egy fontos lépés, mivel szinte minden más dolog is téglalapokkal lesz elintézve. (Kicsit később világosabb lesz.) Ezután blitteljük a téglalapot a kijelzőre, végül frissítünk a szöveg megjelenítéséhez. Na de mi az a blit és miért akarnánk ilyen furcsa nevű dolgot használni? Nos, a kifejezés keletkezése egészen a '70-es évekig nyúlik vissza, és a Xerox PARC-tól jött (aminek sok, ma használt technológiát köszönhetünk). Eredetileg BitBLT volt, ami a Bit (vagy bittérkép) Block Transfer rövidítése. Ebből lett később a Blit (mert így sokkal rövidebb). Gyakorlatilag a képeket és szövegeket a képernyőre dobjuk.

```
# This will make our game window centered in the screen
os.environ['SDL_VIDEO_CENTERED'] = '1'
# Initialize pygame
pygame.init()
# Setup the screen
screen = pygame.display.set_mode((800, 600))
# Set the caption (title bar of the window)
pygame.display.set_caption('Pygame Test #1')
screen.fill(Background)
pygame.display.update()

#  Our Loop
doloop = 1
while doloop:
    if pygame.event.wait().type in (KEYDOWN,
MOUSEBUTTONDOWN):
        break
```

De mi van akkor, ha a szövegünket a képernyő közepére akarnánk igazítani, az első sor helyett, ahol nem annyira van szem előtt? A text.get_rect() és a screen.blit között helyezzük el az alábbi két sort:

```
textRect.centerx =
screen.get_rect().centerx
textRect.centery =
screen.get_rect().centery
```

Itt számoljuk ki, hogy hova kell helyezni az objektumot (ami egy felület), majd a textRect x és y értékét ennek megfelelően állítjuk be.

Futtassuk le a programot. Most már középen van a szöveg, mely a saját felületén helyezkedik el. A szöveget a font.set_bold(True) és/vagy font.set_italic(True) (ebben a példakódban) pygame.font.Font sor utáni elhelyezésével lehet módosítani.

Idézzük fel, hogy amikor egy beépített betűtípust állítottunk be, a „None”-t csak igen röviden tárgyaltuk. Tegyük fel, hogy mi egy valamivel érdekesebb típust szeretnénk használni. Mint már előbb is mondtam, a pygame.font.Font() metódusnak két argumentuma van. Az első az elérési útja és a betűkészlet fájl neve, a második a betű mérete. A probléma, mely-

be beleütközünk többrétegű. Hogyan határozzuk meg annak a betűtípusnak a tényleges elérési útját és fájlnevét, melyet egy adott rendszeren használni szeretnénk? Szerencsére rendelkezik a Pygame egy olyan függvénnyel, ami mindezt elintézi helyettünk, match_font a neve. Itt van egy rövidke kis program, ami kiíratja az elérési útját és fájlnevét (ebben az esetben) a Courier New típusnak:

```
import pygame
from pygame.locals import *
import os
print
pygame.font.match_font('Courier New')
```

Az én rendszeremen a „/usr/share/fonts/truetype/msttcorefonts/cour.ttf” volt a visszaadott érték. Abban az esetben, ha a betűtípus nincs meg, ennek értéke „None”. Ha viszont meglett, akkor ezt az értéket egy változóhoz rendelhetjük és használhatjuk az alábbi kifejezést:

```
courier =
pygame.font.match_font('Courier New')
font =
pygame.font.Font(courier,27)
```

Változtassuk meg a programot úgy, hogy tartalmazza ezeket a sorokat, és futtassuk újra. A lényeg, hogy vagy egy olyan betűtípust használunk, amiről TUDJUK, hogy megtalálható a felhasználó rendszerén, vagy mellékeljük azt a programhoz és belekódoljuk az elérési utat. Más megoldások is lehetségesek, de a megtalálásukat meghagyom számotokra, és most tovább lépünk.

Mindez szép és jó, de grafikákkal még szebb lenne. Találtam egy igen jó tutorialt, melyet Peyton McCollugh írt a Pygame-hez és úgy döntöttem, hogy egy kicsit módosítom. Szükségünk lesz egy képre, ami mászkálni fog a hátterünkön. Ezt a fajta képet sprite-nak nevezzük. Használjuk a GIMP-et vagy valamilyen más hasonló eszközt egy pálcikaember elkészítéséhez. Nem kell nagyon szépet készíteni, egy átlagos pálcikaember is megteszi. Azt fogom feltételezni, hogy GIMP-el dolgozunk. Készítsünk egy új képet, mely 50x50 pixel méretű és az advanced options alatt állítsuk a „Fill With” opciót Transparency-re. Használjuk a

```
import pygame
from pygame.locals import *
import os

Background = 0,255,127
os.environ['SDL_VIDEO_CENTERED'] = '1'
pygame.init()
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption('Pygame Example #4 - Sprite')
screen.fill(Background)
```

ceruza eszközt egy Circle(03)-as ecsettel. Rajzoljuk meg a kis figuránkat és mentsük ugyanabba a mappába, stick.png-ként, amelyben a kódjaink vannak. Itt van az enyém is. Biztosan jobbat is létre tudnátok hozni.

Tudom, nem vagyok egy művész. Ennek ellenére jó lesz arra, amire hivatott. Png-ként mentettük, majd a hátteret átlátszóra állítottuk ahhoz, hogy csak a kis fekete vonalak legyenek majd láthatóak – és nem egy fehér vagy másmilyen háttér.

Beszéljünk most arról, hogy mit fog a programunk csinálni. Egy olyan Pygame ablakot akarunk megjeleníteni, ami tartalmazza a pálcikaembert. Ezen felül szeretnénk, ha mozogna a nyíl billentyűkre – feltéve, ha nem az ablak szélén vagyunk. Az is jó lenne, ha a program kilépne a „q” billentyű megnyomására. Nos, a sprite mozgatása könnyűnek hangzik, és az is, de talán mégsem annyira, amennyire szeretnénk. Két téglalap létrehozásával kezdünk. Az első magáé a sprite-é, a második pedig egy ugyanakkora, de üres kép. Blitteljük a sprite-ot, majd amikor a felhasználó megnyom egy gombot, az üres téglalapot ráblitteljük a sprite-ra, kiszámítjuk az új pozíciót és visszablitteljük azt az új pozicióba. Gyakorlatilag hasonlót csinálunk, mint múltkor az ábécés játékban. És ennyi az egész alkalmazás. Arra elég lesz, hogy megértsük egy kép kirajzolását és mozgatását.

Tehát, hozzunk létre egy új programot és nevezzük el

pygame4.py-nek. Helyezzük el az include-okat, amiket ebben a cikkben használtunk. Ez alkalommal egy menta zöld hátterünk lesz, azaz a színértékek 0, 255 és 127 (lásd előző oldalon).

Ezután létrehozunk egy osztályt, ami lekezeli a grafikát vagy a sprite-ot (itt lenn). Ezt közvetlenül az importok után szúrjuk be.

Mit csinál ez az egész? Nos, kezdjük az __init__ rutinnal. A Pygame sprite modulját a pygame.sprte.Sprite.__init__ sorral inicializáljuk. Ezután beállítjuk a felületet, melyet screennek (képernyő) nevezünk. Ezzel majd meg tudjuk nézni, hogy a sprite lemegy-e a képernyőről. Majd létrehozzuk és beállítjuk az üres oldsprite változót, melynek a fájl nevét (és elérési útját, ha az nem a program mappájában van) adjuk át. Ezután kapunk egy referenciát a sprite-hoz (self.rect), ami automatikusan beállítja a téglalap szélességét és magasságát, illetve az x és y pozícióit a megadott értékekre.

Az update rutin gyakorlatilag csak a sprite-ról készít egy másolatot, majd leellenőrzi, hogy leesne-e a képernyőről. Ha igen, akkor békén hagyja, különben elmozdítja a megadott mértékkel.

```
class Sprite(pygame.sprite.Sprite):
   def __init__(self, position):
        pygame.sprite.Sprite.__init__(self)
        # Save a copy of the screen's rectangle
        self.screen = pygame.display.get_surface().get_rect()
        # Create a variable to store the previous position of the sprite
        self.oldsprite = (0, 0, 0, 0)
        self.image = pygame.image.load('stick3.png')
        self.rect = self.image.get_rect()
        self.rect.x = position[0]
        self.rect.y = position[1]

   def update(self, amount):
        # Make a copy of the current rectangle for use in erasing
        self.oldsprite = self.rect
        # Move the rectangle by the specified amount
        self.rect = self.rect.move(amount)
        # Check to see if we are off the screen
        if self.rect.x < 0:
            self.rect.x = 0
        elif self.rect.x > (self.screen.width - self.rect.width):
            self.rect.x = self.screen.width - self.rect.width
        if self.rect.y < 0:
            self.rect.y = 0
        elif self.rect.y > (self.screen.height - self.rect.height):
            self.rect.y = self.screen.height - self.rect.height
```

A screen.fill utasítás után helyezzük el a következő oldalon (jobbra) látható kódot.

Itt létrehozunk egy példányt az osztályunkból, amit character-nek hívunk. Ezt követően blitteljük a sprite-ot. Létrehozzuk az üres sprite téglalapot és kitöltjük a háttérszínnel, majd frissítjük a felületünket és belépünk a ciklusba.

Addig, amíg a DoLoop 1-el egyenlő, a ciklusban maradunk. A pygame.event.get()-et használjuk a karakterek beolvasásához. Ezután az esemény típusához párosítjuk őket. Ha ez QUIT, akkor kilépünk. Ha KEYDOWN, akkor feldolgozzuk. Megnézzük a kapott karakter értékét és összehasonlítjuk a Pygame-ben definiált konstansokkal. Ezután meghívjuk az osztályunk update rutinját. Figyeljük meg, hogy egyszerűen továbbadjuk a pixelek x- és y-tengelyen való elmozdulásainak listáját. Tíz pixellel töltjük fel (pozitívokkal jobb vagy le esetben, negatívokkal balra és fel esetben). Ha a karakter „q”-val egyenlő, akkor a DoLoopot 0-ra állítjuk, így lépvén ki a ciklusból. Kizárólag ezek után blitteljük a sprite-

unkat az új pozícióba és frissítünk – de ebben az esetben csak a két téglalapot, az üres és aktív sprite-okat, így temérdek időt és számítási kapacitást takarítunk meg.

Mint mindig, a teljes kód a www.thedesignatedgeek.com oldalon, vagy a http://fullcirclemagazine.pastebin.com/DvSpZbaj címen érhető el.

Sok minden van még, amit a Pygame-el meg tudunk oldani. Azt javaslom, hogy látogassuk meg a honlapjukat és keressük meg a referenciaoldalt (http://www.pygame.org/docs/ref/index.html).

Ezen felül még mások által készített játékokat is megtekinthetünk.

```
character = Sprite((screen.get_rect().x, screen.get_rect().y))
screen.blit(character.image, character.rect)

# Create a Surface the size of our character
blank = pygame.Surface((character.rect.width, character.rect.height))
blank.fill(Background)

pygame.display.update()
DoLoop = 1
while DoLoop:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        # Check for movement
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                character.update([-10, 0])
            elif event.key == pygame.K_UP:
                character.update([0, -10])
            elif event.key == pygame.K_RIGHT:
                character.update([10, 0])
            elif event.key == pygame.K_DOWN:
                character.update([0, 10])
            elif event.key == pygame.K_q:
                DoLoop = 0

    # Erase the old position by putting our blank Surface on it
    screen.blit(blank, character.oldsprite)
    # Draw the new position
    screen.blit(character.image, character.rect)
    # Update ONLY the modified areas of the screen
    pygame.display.update([character.oldsprite, character.rect])
```

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

Következő alkalommal egy régi játékommal picit mélyebbre ásunk a Pygame-ben.
Egy NAGYON régi játékommal.

Hogyanok

Programozzunk Pythonban - 16. rész

Írta: Greg Walters

Nemrég megígértem valakinek, hogy megtárgyaljuk a Python 2.x és 3.x verziói közötti különbségeket. Legutóbb pedig azt mondtam, hogy folytatjuk pygame-es programozásunkat. Ennek ellenére úgy érzem, hogy be kell tartanom az ígéretemet, azaz majd csak a legközelebbi alkalommal fogunk a pygame-el foglalkozni.

A Python 3.x verziójában sok változás történt. Minderről sok információ érhető el a Weben, melyek közül néhány linkjét a cikk végén megtalálhatjátok. Ennek ellenére sok aggodalom van a váltás körül. Most csak azokra a különbségekre fogunk összpontosítani, melyek kihatnak az általunk tanultakra.

Vágjunk is bele.

PRINT

Már korábban is utaltam rá, hogy az egyik legnagyobb különbség, a print utasítás használatában van. 2.x alatt egyszerűen írhattuk az alábbit:

```
print “This is a test”
```

és végeztünk is. Ha viszont 3.x alatt próbálkozunk ugyanezzel, akkor a jobbra fenn lévő hibaüzenetet kapjuk.

Ez persze nem jó nekünk. Ahhoz, hogy használhassuk a print utasítást, a kiírandó szöveget az alábbi módon zárójelek közé kell rakni:

```
print(“this is a test”)
```

Nem egy hatalmas változás, de olyan, amire oda kell figyelni. Már azzal fel tudunk készülni a váltásra, ha ezt a szintaxist használjuk 2.x alatt is.

Formázás és változó behelyettesítés

A formázás és a változó behelyettesítés is megváltozott. 2.x alatt a lenti példában látható dolgokat használtuk, illetve a 3.1 alatt is a megfelelő eredményt kapjuk. Azonban ez hamarosan meg fog változni, mert a „%s” és „%d” formázó függvények el fognak tűnni. Az új módszer a „{x}” helyettesítő utasítást fogja használni, melyet a régi példa alatt láthatunk.

Nekem könnyebben olvashatónak tűnik. Továbbá az alábbihoz hasonló dolgokat is tudunk majd csinálni:

```
>>> months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
>>> print "You selected month %s" % months[3]
You selected month Apr
>>>
```

Régi

```
>>> months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
>>> print("You selected month {0}".format(months[3]))
You selected month Apr
>>>
```

Új

```
>>> print "This is a test"
  File "<stdin>", line 1
    print "This is a test"
                         ^
SyntaxError: invalid syntax
>>>
```

```
>>> print("Hello {0}. I'm glad you are here at {1}".format("Fred","MySite.com"))

Hello Fred. I'm glad you are here at MySite.com

>>>
```

Emlékezzünk arra, hogy bár használhatjuk a „%s”-et és változatait, de ezek egy idő múlva teljesen ki fognak veszni a nyelvből.

Számok

Python 2.x alatt, ha ezt írtuk:

```
x = 5/2.0
```

akkor x tartalma 2.5 lett volna. De viszont az

```
x = 5/2
```

kifejezésben a levágás miatt 2 maradt volna. 3.x alatt az:

```
x = 5/2
```

kifejezéssel már 2.5-öt kapunk. A levágáshoz az alábbit kell használni:

```
x = 5//2
```

Input

Régebben volt szó egy, a felhasználó válaszait kezelő, és a raw_input()-ot használó menürendszerről. Valami ilyesmi volt:

```
response = raw_input('Enter a selection -> ')
```

Ez rendben is van 2.x alatt. Viszont 3.x alatt a következőt kapjuk:

```
Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

NameError: name 'raw_input' is not defined
```

Ez nem egy kritikus probléma. A raw_input() metódus az input()-ra lett cserélve. Egyszerűen változtassuk meg a sort:

```
response = input('Enter a selection -> ')
```

és rendben fog működni.

Nem egyenlő

2.x alatt az egyenlőtlenség tesztelésére a „<>” operátort használtuk. Ez azonban már nem megengedett 3.x alatt. Az új változat a „!=”.

Régebbi programok konvertálása Python 3.x alá

A Python 3.x-et egy olyan segédprogrammal kapjuk, ami segít az alkalmazások 2.x-ről 3.x-re való konvertálásában. Ez nem mindig működik, de sok esetben közelebb kerülünk a kívánt végeredményhez. A konverziós programot (logikusan) „2to3”-nek nevezik. Vegyünk egy egyszerű programot. A lenn található példa még régebbről – egészen pontosan a 3. részből – származik.

```
#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)
```

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

Amikor 2.x alatt futtatjuk, az előző oldalon jobbra fent lévő kimenetet kapjuk.

De természetesen, a 3.x-el nem fog működni.

```
File "pprint1.py", line 18
    print
TopOrBottom('=',40)
                   ^
SyntaxError: invalid syntax
```

A problémát a konverziós programmal próbáljuk orvosolni. Mielőtt belevágnánk, érdemes egy biztonsági másolatot létrehozni a konvertálandó alkalamazásról. Én ezt a fájl másolásával oldom meg, úgy hogy a végére biggyesztem a „v3”-at:

```
cp pprint1.py pprint1v3.py
```

Több fajta módszer van a program futtatására. A legegyszerűbb, ha megkerestetjük a programmal a hibákat. Ezt balra lent láthatjuk.

Figyeljük meg, hogy az eredeti forráskód nem változott. A változások fájlba való mentéséhez a „-w” paramétert kell használnunk. Ennek eredményét jobbra lenn találjuk.

Azt is észrevehetjük, hogy a kimenetek megegyeznek. Ez alkalommal a forrásfájlunk

```
> 2to3 pprint1v3.py
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
RefactoringTool: Refactored pprint1v3.py
--- pprint1v3.py (original)
+++ pprint1v3.py (refactored)
@@ -15,9 +15,9 @@
 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)
+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))
RefactoringTool: Files that need to be modified:
RefactoringTool: pprint1v3.py
```

```
> 2to3 -w pprint1v3.py
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
RefactoringTool: Refactored pprint1v3.py
--- pprint1v3.py (original)
+++ pprint1v3.py (refactored)
@@ -15,9 +15,9 @@
 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)
+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))
RefactoringTool: Files that were modified:
RefactoringTool: pprint1v3.py
```

viszont „version 3.x compatible” fájllá változott (jobbra).

A program most már működik 3.x alatt is, valamint, mivel egy elég egyszerű programról van szó, a 2.x verzióval is kompatibilis maradt.

```
#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))
```

Most azonnal váltsak 3.x-re?

A legtöbb probléma a többi nyelv változásánál is előfordul. A szintaxis átalakul minden egyes új verzióval. A rövidítések, mint += és -= a semmiből hullanak elénk, hogy megkönnyítsék életünket.

Hogy mekkora a hátránya a 3.x-re való migrálásnak? Nos, van egy kicsi. Sok olyan függvénykönyvtár van, ami még 3.x alá nem érhető el. Olyan dolgok, mint például a Mutagen – melyet néhány számmal korábban már használtunk – sem használható még. Annak ellenére, hogy mindezek fogós érvek, nem jelentik azt, hogy fel kell adnunk a Python v3.x-et.

Az én javaslatom az lenne, hogy kezdjünk el a helyes 3.x-ás szintaxissal kódolni. A Python 2.6 támogat majdnem minden olyan dolgot, melyet 3.x-ben használnánk. Ezzel a módszerrel készen fogunk állni a 3.x-re való váltáshoz, amikor annak el fog jönni az ideje. Ha viszont meg tudunk lenni a szabványos modulokkal, akkor akár bele is vághatunk. Másrészt, ha a határokat akarnánk feszegetni, érdemes addig várni, amíg a modulok fel nem zárkóznak (mert biztosan fel fognak).

Lentebb pár linket találunk, amikről úgy gondoltam, hogy hasznosak lehetnek. A legelső a 2to3 használati útmutatója. A második egy 4 oldalas puska, amit elég jó referenciának találtam. A harmadik pedig egy olyan könyv a Pythonról, amit a legjobbnak tartok. (Legalábbis addig, amíg meg nem írom a sajátomat.)

Találkozzunk legközelebb is!

Linkek

A 2to3 használata:
http://docs.python.org/library/2to3.html

Moving from Python 2 to Python 3 (A 4 oldalas puska)
http://ptgmedia.pearsoncmg.com/imprint_downloads/informit/promotions/python/python2python3.pdf

Dive into Python 3
http://diveintopython3.org/

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
Kovács Roland Attila
Tömösközi Máté Ferenc

Szerkesztő, korrektor:
Heim Tibor

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