Alkalmi Python, 3. rész

PCLinuxOS Magazine - 2019. április

Írta: Peter Kelly (critter)

Dolgok

A python tanulása közben elolvasott sok könyv közül az egyik Mark Lutz „Learning Python”-ja volt. A könyvben azt írta, hogy a „programok dolgokat tesznek a dologgal”. Így összegzi és hozza le a mi szintünkre. Nos, mi a „dolog” és mik a „dolgok”? A teljes válasz olyasmi, hogy „bármit bármivel”, egyelőre elégedjünk meg annyival, hogy szokásos dolgokat teszünk a dolog lényegének könnyebb megértése érdekében.

A Python „dolgot” (anyagot) objektumnak (object) hívja és a Python számos fajtát ismer. Valójában a Python-ban minden objektum. Ahhoz, hogy ezekkel az objektumokkal dolgozzon a Python, általában függvényeket és eljárásokat használ. Számos objektumtípus van, és van néhány alapvető objektumtípus, amit valóban ismernünk kell, ha tovább akarunk lépni. Minden objektumtípusnak megvan a saját adag eljárása, amiket szintén ismernünk kell.

Az alap típusokat így lehetne felsorolni:

- számok
- sztringek (karaktersorok)
- listák
- vektorok (tuple)
- szótárak
- tömbök
- fájlok

Valószínűleg több van, de van némi átfedés közöttük, mint a számok és a boolean-ok között, vagyis a megkülönböztetés zavaros. A következő projektünk sztringeket, listákat és fájlokat használ, ezért az alapjaikat tisztázzuk. A számokkal már foglalkoztunk.

Sztringek

A sztring egy karaktersorozat. A karakterek lehetnek alfanumerikusak, írásjelek, szóközök, vagy bármi, amit az utf-8 szabvány lefed. Ne foglalkozz túl sokat ez utóbbival. Ha le tudod írni, akkor az utf-8 tartalmazza, a számítógép így tudja, hogyan kell megjeleníteni. Az összes programnyelv használ sztringeket. Néhány, mint a C, rendelkezik karater típussal is, ami egyetlen karaktert jelent. Pythonban nincs ilyen. A Pythoban ez 1 hosszúságú sztring. A Python sztringjei tetszőleges hosszúságúak lehetnek, ami csak a memóriába belefér (ám ha valóban hosszú sztringre van szükséged gondolkodj fájlban).

A sztring Pythonban str típusú objektum és néhány más objektum konvertálható sztringgé az str() függvény segítségével.

```
>>> str(11.75)
'11.75'      # a lebegőpontos típus sztringgé lett átalakítva.
```

A sztringek „megváltoztathatatlanok”, ami annyit tesz, hogy ellenállnak a változtatásnak, Ez nagy hátránynak tűnhet, de valójában, ahogy látni fogod, épp ellenkezőleg, és vannak módszerek új sztringek készítésére.

Sztringre általában változó hozzákötésével hivatkozunk, ami tényleges névadást jelent.

```
s = 'The python language'
```

Ahányszor csak a Python átadja az s változót, az azonnal a helyettesített szövegre cserélődik. Mivel a sztringek változtathatatlanok, ezért biztos lehetsz a jelentésében. A programnak nincsen olyan része, ami megváltoztathatná. A sztringek darabolhatóak a szeletelő (slicing) eljárással, de az eredeti változatlan marad. A megőrzendő a szeleteket új névvel kell ellátni (változóhoz kötni). A szeletelést szögletes zárójel hajtja végre, ami tartalmazza az induló pozíciót, a záró pozíciót és az opcionális ugrást, vagy „léptetést”. A záró karaktert nem veszi át. Ha nincs nyitó, vagy záró megjelölés, akkor az elejét, vagy végét elveszi. A karakterek indexelése 0-ról indul, vagyis ez van:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
T h e   p y t h o n     l a n g u a g e
18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

```
s[0:6] ==> 'The py'
s[4:10] ==> 'python'
s[:10] ==> 'The python'
s[:10:2] ==> 'Tepto'    # legelejétől indul a 10.-ig kizárólag 2 lépésenként.
s[:] ==> 'The python language'
```

Ez utóbbi gyors módja másolat készítésének.

Negatív értékek visszafelé számolnak 1-től (nem lehet mínusz 0):

s[-3:] ==> 'age'
s[-12:-4] ==> 'hon lang'
s[:-9] ==> 'The python'

Az s továbbra is a „The python language” hivatkozása, Nem lehet megváltoztatni. Hozzá kell szokni, változtathatatlan, vagyis megmarad.

t = s

Mind az s, mind a t ugyanarra a sztringre hivatkozás a 'The python language'-re.

Ahogy a dolgokat tesszük a sztringekkel azok az eljárások és szinte mindenre, amit csak tenni akarhatsz sztringekkel, létezik eljárás. Az eljárásokat „pont jelöléssel” hívjuk meg.

s.upper() ==> 'THE PYTHON LANGUAGE' # nagybetű
s.find('python') ==> 4, a beágyazott 'python' sztring első betűjének pozíciója
s.replace('python', 'English') ==> 'The English language' # csere
s.split() sztringek listájával tér vissza ==> ['The', 'python', 'language']

Ezekből az eljárásokból nagyon sok van, amit a hivatalos dokumentáció tartalmaz https://docs.python.org/3/, ám jó indulás lehet C H Swaroop korábban említett könyvének elolvasása: https://python.swaroopch.com/.

Amikor a számokkal foglalkoztunk, bemutattam a python-fordító használatát. Ugyanakkor van egy jobb változat, ami a tárolókban telepítésre elérhető, ipython a neve. Ez egy közvetlen helyettesítője a szabvány fordítónak, de sokkal több funkcióval rendelkezik. Az első különbség, amit észrevehetsz, hogy a prompt megváltozott >>> -ról ***'In [1]:***' -re, és színes, alapbeállítás szerint zöld. Amikor beírsz egy kifejezést és lenyomod az Entert egy új sor jelenik meg, ami ***'Out [1]:'*** -vel kezdődik és alapból piros és ezt követi a kifejezés kimenete. Minden alkalommal, amikor új kifejezést viszel be a zárójelben lévő szám eggyel növekszik. Ez a parancstörténet. Egy jó csomó egyéb előnye van ezen a fordító használatának, ezek közül néhánnyal foglalkozunk hamarosan.

A dolgokkal munkálkodást általában „adatfeldogozásként” említik, és amikor a dolog szöveg, akkor „szövegfeldolgozásként”. Ám, ahogy azt tisztelt Mr. Lutz jelezte, továbbra is dolgokkal csinálunk dolgokat. Noha a sztring nem változtatható, az objektum, ami az s változóra hivatkozik igen, és bármilyen objektumtípusra változtatható.

s = s[4:10] ==> 'python'
Most az s új sztringre a „python”-ra hivatkozás, amit szeleteléssel nyertünk. Az eredeti sztring érintetlen, így bármi, ami arra hivatkozott szintén, mint például a t, szintén érintetlen maradt. Ha nem készül további hivatkozás erre, akkor automatikusan el lesz távolítva a memóriából egy „szemétgyűjtésnek” nevezett folyamatban.

s = 3.14159
Az s most egy lebegőpontos (decimális) szám.
t ==> 'The python language' . # t változatlan

A sztringeket lehet összefűzni és az ismétnit összeadás és szorzáson útján.

t + ' is powerful' ==> 'The python language is powerful'
t[4:10] * 3 ==> 'pythonpythonpython'

Ez kombinálható

(t[4:10] + ' ') * 3 ==> 'python python python ' # szóközök kerültek bele.

A sztringben lévő karaktereket idézőjelbe kell tenni, ez akár egyes, akár kettős lehet. Nem számít, melyiket használod, ám ha a sztringben van idézőjel, akkor a másik típust kell használni.

'He said "Hello" to the stranger'
"Python's syntaxis"

Létezik egy harmadik típusú idézőjel Pythonban, triplának hívják. Használható egyes, vagy kettős, és lehetővé teszi több sor áthidalását. Ezeket elterjedten használják a kódok dokumentációjában.

""" Ez a függvén egy szám négyzetét számítja ki.
sq(x)
x a négyzetre emelendő szám
az x négyzetét adja vissza """

A fordító használata – én itt az ipython-t használom – a következőket gépeld be:

In [2]: dir(str)
Out[2]:
['__add__',
'__class__',
'__contains__',
...
... egy csomó eljárást kitöröltem
...
'rstrip',

'split',
'splitlines',
'startswith',
'strip',
'swapcase',
'title',
'translate',
'upper',
'Zfill']

Így kapsz egy teljes listát a sztringekre elérhető eljárásokról. Egyelőre ne foglalkozz a dupla alsó aláhúzással kezdődőkkel. Ha például azt írod be, hogy:

In [3]: help(str.s

Lenyomod az Entert és az s kezdetű eljárások neveit jeleníti meg

In [3]: help(str.s
str.split str.strip
str.splitlines str.swapcase
Str.startswith

A nyilakkal válassz ki egyet, zárd be a zárójelet és nyomj Entert.

In [3]: help(str.strip)

Help on method_descriptor:

strip(self, chars=None, /)

A sztring egy másolatát adja úgy, hogy az indító és a záró szóközt törölte. Ha a karakter meghatározása nem None (nincs), akkor a megadott karaktert távolítja el. Ez alapszintű súgó. Nyomj 'q'-t a visszatéréhez. Próbáld ki.

Ez egy alapszintű súgó az eljárásról. Nyomj „q”-t a kilépéshez. Próbáld ki.

In [4]: s = ' the end '
In [5]: s.strip()
Out[5]: 'the end'

A tabulátoros kiegészítés nagy segítség a begépelésénél. A színkódolás a különféle objektumtípusokra kiterjed, ami szintén segít. Az előzményeket az egyes futtatások között menti, a változók értékeit nem. Nyomj fel nyilat a visszalépéshez.

Listák

Pythonban egy lista „dolgok” rendezett gyűjteménye, amiket szögletes zárójelek tartalmaznak és vesszők tagolják. A „dolgok” lehetnek bármilyen típusú objektumok, akár más listák is. A sztringgel ellentétben a listák elérhetőek, bárhol megváltoztathatóak.

```
list1 = ['The', 'python', 'programming', 'language']       # egy sztring-lista
list2 = [42, 3.14159]      # két típusú számok listája
list3 = ['Linux', 17, 42]  # kevert lista
list4 = []           # üres lista
list5 = list1        # a list1 és a  list5 azonos
# list
list1 == list5 ==> True   # a két lista azonos
list1 is list5 ==> True    # egyazon objektumra hivatkoznak
```

A sima „=” az értékadásra való, a „==” az egyenlőség tesztelésére.

A sztringekhez hasonlóan a listák is szeletelhetőek.

list6 = list1[:] # list6 a list1 másolata
list6 == list1 ==> True # (igaz) azonos értékűek
list6 is list1 ==> False # (hamis) eltérő objektum referencia

Csakúgy, mint a sztringeknél, összefűzés és ismétlés támogatott. Ezen műveletek eredménye egy új, önálló lista, amit egy változóhoz kell kötni, hogy megmaradjon. Az eredeti listák változatlanul maradnak. Az ismétlés szorzószáma egész kell legyen.

list1 = [7, 3]

In [6]: list1 + list1
Out[6]: [7, 3, 7, 3]

In [7]: list1 * 3
Out[7]: [7, 3, 7, 3, 7, 3]

In [8]: list1
Out[8]: [7, 3]

In [9]: list1 * list1

TypeError Traceback (most recent call last)
<ipython-input-9-86e343e4b412> in <module>
----> 1 list1 * list1
TypeError: nem lehet nem integer típusú „listával” szorozni.

A listák indexelhetőek is.

list1[1] ==> 'python'
list3[2] ==> 42

Az elemek „pattinthatóak” (pop) # a végéről eltávolíthatóak.

list1.pop(2) ==> 'programming'
list1.pop() ==> 'language'
list1 most már ['The', 'python']

Törölhetőek is. (clear)

list3.clear() ==> []

Elemeket lehet hozzáadni. (append)

list2.append(1.6e3) ==> [42, 3.14159, 1600.0]
 # 1.6e3 is 1.6 * 103

Vagy az index előtti pozíció elé beilleszthetőek. (insert)

list2.insert(2, 4096) ===> [42, 3.14159, 4096, 1600.0]

És rendezhetőek. (sort)

list2.sort() ==> [3.14159, 42, 1600.0, 4096]
list1.sort() ==> ['The', 'language', 'programming', 'python']

A listák változtathatóak, hatással lesz az összes változóra, ami hivatkozik rá.

list5 ==> ['The', 'language', 'programming', 'python']

A megváltoztathatatlan objektum nem is annyira rossz dolog!

A rendezés kulcs felhasználásával kontrollálható.

list6 ==> ['The', 'python', 'programming', 'language']
list6.sort(key=str.lower) ==> ['language', 'programming', 'python', 'The'] # kisbetűs = lower

Itt a lower eljárást, ami a sztring kisbetűs változatát adja vissza, rendező kulcsnak használjuk.
Ez a lista objektumok saját rendezési eljárását használja, de van sorted nevű beépített rendező is (az ilyen funkciókat nem meglepő módon „beépítettnek” hívják). Ezt akkor lehet használni, amikor az objektumtípusnak nincs saját eljárása. A rendezett objektumot nem frissíti automatikusan és kézzel kell változónévhez kötni. A rendezendő elemek azonos típusúak legyenek, különben hibajelzést fogsz kapni.

list5 = [99, 17, 42]
list5 = sorted(list5) ==> [17, 42, 99]

Ugyanakkor

list6 = [17, 'Linux', 9]
list6 = sorted(list6)
Traceback (most recent call last):
File "list_sort.py", line 13, in <module>
list6 = sorted(list6)
TypeError: '<' not supported between instances of 'str' and 'int'

Ilyen hibajelzések tömegét fogod kapni, amikor először saját kódot írsz. Ahogy összetettebb dolgokra térünk át, megmutatot a hibakeresés keressük és kezelés módját – ez olyan, amit még a gyakorlott programozók is tesznek, ne keseredj el.

Fájlok

A fájlok az olyan „dolgok” állandó tárolására szolgálnak, amikkel „dolgokat” teszünk. A tárolás általában helyi, vagy távoli merevlemezen történik. A távoli tárolás tipikusan a hálózatos gépekre jellemző, legyen az web alapú, vagy felhő tároló, de a python szempontjából kicsi a különbség.

Egy fájl használatához ismerni kell a helyét és megfelelő hozzáférési joggal kell rendelkezni fájl, vagy könyvtár olvasáshoz, íráshoz, létrehozáshoz, vagy törléshez. A fájlt a tervezett művelethez megfelelő módon kell megnyitni, és számos mód áll rendelkezésre:

'r' olvasási mód, ha nincs jelölve, ez az alap

'w' írás.

'a' hozzáfűzés

'b' bináris, vagy byte fájlok.

't' szöveges fájlokra az alap, de az elhagyása feltételezett, így ritkán látható. A 'wt' szöveges módú írást jelent.
Egy „+” hozzáadva pl. „r+” író olvasó módot jelent, és ha a mód „w”, vagy „a”, akkor nem létező fájl esetén létrehozza azt.

Óvatosan! A „w” write felülír bármi is legyen ott. Ha nem szeretnéd, akkor használd az „a” append-et. „b” nélkül a fájlokat szöveges módban nyitja meg. Általában ezt akarjuk, de ha bináris adatokkal dolgozol, pl. grafika, vagy hang, akkor a fájlt byte módban kell megnyitni. A kettő nem felcserélhető és a rossz mód használata legjobb esetben is szemetet produkál, rosszabb esetben adatvesztést.

Most maradjunk text módban. Amikor végeztél a fájllal, be kell azt zárni. Általá-ban a python bezár minden nyitott fájlt, amikor a program bezáródik. Ám ha a program összeomlik, amikor a fájl még nyitva, nagy esély van adatvesztésre. É-letünk megkönnyítésére a python rendelkezik valami context managernek hívott valamivel, ami biztosítja, hogy a fájlok még összeomlás esetén is bezáródjanak.

Ha valami ilyet csinálunk:

```
with open('myfile.txt', mode='a+') as f:
f.write( 'Adding to file...\n')
```

A write nem feltétlenül teszi ki a sorvégzáró jelet (\n), tehát nekünk kell megtenni. Ezután futtasd ezt a kódot 5-ször, és ha azután futtatjuk ezt a kódot:

```
with open('myfile.txt', mode='r') as f:
print(f.read())
```

Ezt látjuk:

```
Adding to file...
Adding to file...
Adding to file...
Adding to file...
Adding to file...
```

Alternatívaként használhatjuk a fájlobjektumok sorolvasási eljárását egyetlen sor olvasásához.

```
with open('myfile.txt', mode='r') as f:
print(f.readline(), end='')
```

```
adding to file...
```

A print kifejezésnél az end='' szükséges, mivel a print funkció alapból új sor karaktert **ad** hozzá (ahhoz, amit a fájl írásakor mi adtunk hozzá), ahogy azt láthatjuk, ha újra végrehajtjuk a fájlon:

```
with open('myfile.txt', mode='r') as f:
    for line in f:
        print(line)
```

```
adding to file...

adding to file...

adding to file...

adding to file...

adding to file...
```

Ez furcsának tűnhet, de amikor egy fájlba írunk, pontosan meg akarjuk határozni, hogy mi kerüljön a fájlba. Ha valamit kinyomtatunk, általában új sort akarunk, de szükség esetén felülírhatjuk ezt a end= záradékkal. Amikor korábban alkalmaztuk a print(f.read ())-et, a nyomtatási funkció kimenete került a fájlba, majd ehhez adta hozzá a végső új sort.

De mi van, ha ....?

Ha egy parancsfájl egyszerre csak egy sort tudott végrehajtani a megjelenés sorrendjében, akkor szigorúan be lennénk határolva. Néha döntéseket kell hozni, és a döntés eredményétől függően kell más és más intézkedést tenni. Alkalmanként ugyanazt a „dolgot” kell tenni egy jó csomó „dologgal”. Ezt hurkolásnak és elágazásnak, vagy programvezérlésnek nevezik.

A döntéshozatal az **if** és az opcionális **elif**, valamint **else** kifejezéssel vezérelhető. Ez a következő formában történik

```
if condition is true: # ha a feltétel igaz
    do things to stuff # csináld ezt a dologgal

elif a different condition is true: # ha egy másik feltétel teljesül
    do other things to stuff # más dolgokat csinálj a dologgal

Else: # más esetben
    do this # ezt csináld
```

Van még a **while** kifejezés, ami valahogy így néz ki:

```
while condition is true: # ameddig a feltétel teljesül
    do things to stuff # csináld ezt a dologgal
```

Ezzel a tevékenység a dolgokkal addig folytatódik, amíg a feltétel teljesül.

Van még a **break**, ami egyenesen kiugrik a hurokból **continue** kifejezés, ami megszakítja a hurok aktuális ismétlését és a következő lépéssel folytatja.

Arra, hogy dolgok csoportjával csináljunk dolgokat, a **for** kifejezés szolgál, amit a következő példában használtunk ismétlésre fájlokon.

for all of stuff in this collection of stuff: # a gyüjtemény összes elemére
do these things # ezeket hajtsd végre

Figyeld meg ezeket a következő alkalmazás kódjában, amiben elkezdjük használni ezeket az új struktúrákat.

Mind a szabványos, mind az ipython fordítóban van egy három pontból álló, folytatást jelző prompt az olyan többsoros kifejezésekre, mint a hurkok.

```
In [6]: for i in s.strip():
...: if i == ' ':
...: print('Found a space')
...:

Found a space
```

Mára elég az elméletből!

Docz átalakítva

Eddig nem használtunk túl sok kódot, mert nem igazán ismertünk sokat. Ez hamarosan megváltozik, mint a következő alkalmazásban, ahol a most tárgyalt elmélet egy részét használni kezdjük. Az alkalmazás nem csak parancssori eszközt használ egy funkció végrehajtásához. Némi fájlkezelés és szövegfeldolgozás lesz egér- és billentyűzetérzékeny GUI-ban. És valójában nagyon hasznos.

Azt szeretnénk, hogy ez az alkalmazás nem csupán néhány fájl nevét jelenítené meg. Ahhoz, hogy hasznos legyen, el kell indítania a kapcsolódó alkalmazást, hogy dolgozhassunk rajta. Ebben az esetben az indítandó alkalmazás a LibreOffice Writer, amihez ez a parancs kell

Libreoffice --writer filename.odt Ahol a fájlnév.odt-t a docz alkalmazásból szerezzük.

A könnyebbség kedvéért belépünk a fájlokat tároló könyvtárba. Ehhez egy másik, az os nevű szabványos modult kell importálnunk, amivel néhány hasznos operációs rendszerparancsra teszünk szert. Az általunk keresett eljárások az os.path.expanduser () és az os.chdir (). Az első kifejti a '~' karakterláncot a felhasználó saját könyvtárának elérési útjára, a második könyvtárat vált. Azt is meg kell mondanunk az alkalmazásnak, hogy mit kell tennie, ha az egérrel fájlnévre kattintunk (és mit kell tenni, ha a fájllistát követő üres helyre kattintunk).

Hogy az alkalmazás tudja, hogy hol kattintottunk, kell egy kurzor, amit a Qt ad és egy másik Qt importáló kifejezés is, mivel az a két importáló, amink van erre nem képes. Ez elég fejlett dolog, de akkor is használhatjuk, ha nem is igazán értjük.
A módosított kód így néz ki.

```
#!/usr/bin/env python3

import sys

from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
import subprocess
import os
import docz_ui

class Docz(QWidget, docz_ui.Ui_Form):
    def __init__(self):
        super(self.__class__, self).__init__()
        self.setupUi(self)
        self.quitButton.clicked.connect(self.exitApplication)
        home = os.path.expanduser('~')
        os.chdir(home + '/Documents/wills_plays/')
        cmd = "ls -1 *.odt" # that's -one not -ell
        result = subprocess.check_output(cmd, shell=True).decode('utf-8')
        self.resultslist.setText(result)
        self.resultslist.mousePressEvent = self.mousePressed

    def mousePressed(self, Event):
        textCursor = self.resultslist.cursorForPosition(Event.pos())
        textCursor.select(QTextCursor.BlockUnderCursor)
        self.resultslist.setTextCursor(textCursor)
        f_name = textCursor.selectedText()
        if f_name == '': # mouse was clicked in empty space
            pass         # so ignore it
        else:
            f_name = f_name[1:] # drop the first character
            command = ('libreoffice --writer ' + f_name)
            subprocess.Popen(command, shell=True)
            self.exitApplication()
    def keyPressEvent(self, e):
        if e.key() == Qt.Key_Escape:
```

```
        self.exitApplication()

    def exitApplication(self):
        self.close()
        sys.exit()

if __name__ == '__main__':
app = QApplication(sys.argv)
form = Docz()
form.show()
app.exec_()
```

Ha a látható kódot futtatod, gondoskodj arról, hogy legyen néhány writer-fájl a home könyvtáradban, amit az alkalmazás megtalálhat.

Most mit csináltunk? Hozzáadtunk az indító kódhoz egy sort, ami egy egérkattintást egy új mousePressed eljáráshoz kapcsol. Az eljárás első négy sora beállít egy QtextCursor-t, a Qt így határozza meg a pozíciót a szövegben. Szintén ebben az eljárásban van egy if - else annak vizsgálatára, hogy az egér lenyomása szöveg, vagy üres résznél volt-e. A kapott fájlnév első karaktere egy láthatatlan, bekezdést elválasztó elem, ami nem kell. Ezért létrehozunk egy f_name változót egy olyan szelettel, ami nem tartalmazza azt. Ez az, ahogy vágjuk a sztringeket és áthelyezzük, mivel a sztringek nem változhatnak, érintethetetlenek.

if a szöveg üres – pass. A pass a python ne csinálj semmit parancsa, ami ezt teszi – semmit!

else összerak egy parancsot a hivatkozott szövegből

végrehajtja a parancsot és bezárja az alkalmazást.

Végül nyisd meg a Designer-t és a textedit betűméretét 12-ről 14-re váltsd. Így könnyebb a megfelelő fájl kijelölése. Mentsd és futtasd az update_res.sh-t.

Azt meg kell itt jegyeznem, noha van egy alap keyPressedEvent eljárás, mi a Qt textedit-jének a mousePressEvent-jét használjuk, így más nevet kell adnunk a kattintást érzékelő rutinunknak, lkerülendő az ütközést a mousePressed-del.