Alkalmi Python, 1. rész

PCLinuxOS Magazine - 2016. február

Írta: Peter Kelly (critter)

Bevezetés

Egy programnyelv használatának tanulása hosszú, bonyolult és időnként frusztráló folyamat. A legtöbb dokumentumot és könyvet gyakorlott programozók és számítógépesek írják, kifejtve a nyelv minden aspektusát részletesen. Amennyiben hivatásos programozó akarsz lenni, akkor ezen az úton kell járni. De mi legyen velünk, többiekkel? Az egyszeri felhasználóval és amatőrrel, aki képes akar lenni kicsi, grafikus felületű hasznos alkalmazások készítésére saját maguknak, vagy korlátozott használatra.

Nem vagyok szakértő és semmiképpen sem elég képzett, hogy megmondjam, hogyan programozz Python-ban, de talán segíthetek az első lépéseket megtenni a nyelvben. Egyike vagyok az előbb említett alkalmi programozóknak, innen ennek az írásnak a címe.

Egy új programot először általában egy hello_world programmal szoktak bemutatni, ami egyszerűen kinyomtatja a terminálra, hogy „Hello world!”. Python3-nál ez így nézne ki:

```
print("Hello world!")
```

Esetünkben ez nem túl hasznos. Ezután unalmas és ismeretlen kifejezéseket, illetve a szintaxist kellene megtanulnod, szerkesztőbe begépelni némi kódot, majd végrehajtani azt terminálban. Szintén szokásos, hogy nem fut és meg kell vizsgálni, próbálva megtalálni, mi romlott el. Alkalmasint el tudod indítani és a munkád eredménye az lesz, hogy valami új szöveg jelenik meg a terminálban – csodás! Ennyi? „Nem”, kapod a választ. „De uralnod kell ezt, mielőtt elkezdenél írni valódi, grafikus alkalmazásokat.” Nem meglepő, hogy ez sokakat elriaszt.

A problémánk az, hogy mi grafikus alkalmazást akarunk készíteni, méghozzá most, de ehhez a nyelv nagy részét ismernünk kell, valamint néhány elég bonyolult és fejlett programozási technikát is. Úgy tűnik, egyfajta tyúk és tojás helyzet van. Szerencsére van kerülő út, de ehhez történetesen tanulnod kell egy kicsit a Python-ról, ha egy kicsit is ismered a Python nyelvet, az segít. A nyelvhez kiváló bevezető ingyenes könyv **C. H. Swaroop** „A Byte of Python” című műve.

Ez egy kezdőknek szánt szélvész gyors utazás a nyelvben, néhány este alatt át lehet futni. Éppen elég ahhoz, hogy meghozza az étvágyad. Olvasd el. Nem időpocsékolás.

Bemutatom és megismertetem majd a Python nyelv néhány elemét, ahogy azok a példaalkalmazásokban megjelennek.

Programozást tanulva az egyik oktatott módszer az úgynevezett „fentről lefelé tervezés”. Úgy működik, hogy a végeredménnyel indítunk és onnan haladunk visszafelé, hozzáadva utasításokat, amik a végeredményt produkálják. Például:

New price = $21 # (Új ár) Az eredmény, amit akarunk.

Az új ár a régi ár plusz a csökkentés, vagy növelés kell legyen, valahogy így:

New price = Old price + Price change # *(régi ár + változás)*
New price = $21

A régi árat megkereshetjük, de az árváltozás a régi ár valamilyen százaléka lehet:

Old price = $20
Percentage = 5% # *(százalék)*
Price change = Old price x Percentage # *(Árváltozás = Régi ár x Százalék)*
New price = Old price + Price change
New price = $21

Így a program struktúrája kialakult. Ez nem valamilyen programnyelven íródott, hanem egy pszeudokódnak nevezett valamivel, ami azt jelenti, hogy megtehető anélkül, hogy előtte megtanulnák egy nyelvet. Egyszerűen írj le bármit, ami értelmes és a műveleteket azonosítja. Amikor új projektbe kezdek, nemcsak azt nem tudom, hogy a program hogyan épül fel, de gyakran azt sem, hogy hogyan írjam meg a kódot, ami elvégzi, amit akarok, illetve hogyan szerezzem meg az adatot, szöveget a végrehajtáshoz.

A program ilyetén bontása megadja a kiindulópontot és a hiányzó részeket, amit, ahogy haladok, töltöm fel. Az Internet csodás hely arra, hogy megtudd, miképpen kell a dolgokat csinálni. Bármibe kezdenél biztos lehetsz, hogy más már próbált valami hasonlót csinálni és a kérdéseire olyan oldalakon, mint a stack exchange már kapott választ. Ha még senki sem próbálta volna azt, amit te akarsz, kérdezd meg magadtól, van-e más, könnyebb mód a cél elérésére.

A „Rapid Application Development” (gyors alkalmazásfejlesztés) bevezetése olyan másik áttörés a kilencvenes évekből, ami elérhetőbbé tette a programozást a „hétköznapi” emberek számára pl. az MS Visual Basic és Borland Delphi segítségével. Ezekkel a rendszerekkel a felhasználók grafikus felületeket készíthettek gombok és dolgok húzásával majd ledobásával egy űrlapra, majd némi kód hozzáadásával a kívánt működésre bírhatták.

Amennyire tudom, (még) nincs olyasmi, hogy „Visual Python”, de közelítünk hozzá. A legegyszerűbb grafikus rendszer, amit Python-nal használni lehet az a „tkinter”, de a Qt5 sokkal többet nyújt, ezért én ezt

használom. A mostani KDE/Plasma asztal is Qt5-re épül. A Qt5 rendelkezik még egy nagyon hasznos eszközzel, a Designer-rel, ami a húzás és dobás részt kezeli számunkra. Használatával készítek egy nagyon egyszerű alkalmazást, aminek gyakorlati haszna nincs, de működik. Lesz saját ablaka, néhány gombja és terület szöveg számára. Átméretezheted, lecsukhatod és mindent megtehetsz vele, amit a grafikus alkalmazásokkal szokás. Mindezt anélkül, hogy bármilyen programozási nyelvet meg kellene tanulnod. Ez lesz a sablon alkalmazásunk, amihez funkciókat adhatunk a nyelv tanulása során.

Kezdjünk bele

Először is a Python programnyelvre lesz szükség. Jelenleg két verzió érhető el belőle: a python2 és a python3. Nagyon hasonlítanak egymásra, de nem teljesen kompatibilisek. A python2 élettartama végét 2020-ban éri el, ezért a python3-at fogom használni (ami szerintem jobb is). A python3-at a PCLinuxOS alapból telepíti.

Ezután szükségünk lesz a Python kapcsolódására egy grafikus eszközkészletkönyvtárhoz. Ha nem tudod, hogy ez mit jelent, ne foglalkozz vele! Egyszerűen folytasd. Synaptic-ból telepítsd ezeket:

python-qt5
python3-pyqt5-sip
python3-qt5
python3-sip
és qttools5-designer

A Qt5 a Qt eszközkészlet legfrissebb verziója.

Kell még egy szerkesztő is. Bármilyen sima szövegszerkesztő jó, de én a geany-t használom, ami szintén megtalálható a tárolóban és telepíteni kell. Ha geany-t használsz, könnyebben követheted a cikket.

Végül csinálj egy **pyuic5** nevű fájlt a következő tartalommal:

```
#!/bin/sh
exec /usr/bin/python3 -m PyQt5.uic.pyuic ${1+"$@"}
```

És egy másik pyrcc5 nevűt ezzel a tartalommal:

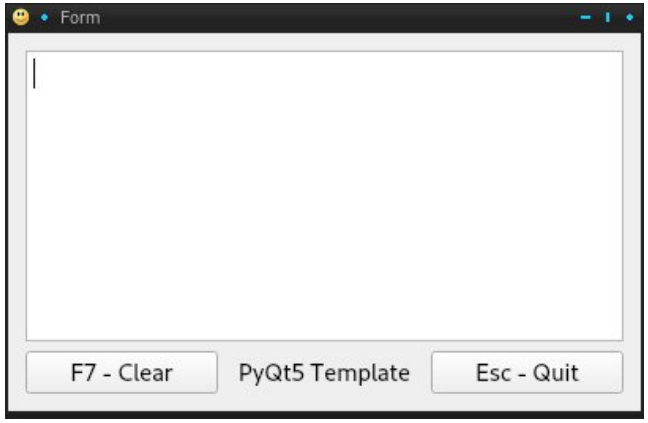
```
#!/bin/sh
exec /usr/bin/python3 -m PyQt5.pyrcc_main ${1+"$@"}
```

Root-ként másold azokat a /usr/bin-be.

OK. Ha KDE Plasma-t használsz, rendben is vagyunk. Ha másfajta asztalt használnál, vagy a könnyebb darkstar kiadást, akkor további telepíteni valód is van. A Python hibaüzenetei megmondják, hogy mi hiányzik.

A sablon

A sablonalkalmazás, amivel kezdünk nagyon egyszerű (és elég fölösleges), de működik és a további ambiciózus projektjeink alapját jelenti. Valahogy így néz ki:



Nem sok minden látható rajta és csak annyit enged, hogy a mezőbe írhatsz és azt a Clear gomb, vagy az F7 billentyű lenyomásával törölheted. Az alkalmazást az Esc, a Quit, vagy az ablak jobb felső sarkában a close gomb lenyomása bezárja. Rendelkezik a maximalizálás, minimalizálás és a megragadással való mozgatás képességével, illetve az átméretezés lehetőségével egyik oldalát, vagy sarkát megragadva, amire az ablakon belüli dolgok, a gombok stb. automatikusan áthelyeződnek, de még saját ikonja is van. Mindez annyit tesz, hogy az alkalmazás már rendelkezik sok mindennel, amit egy „megfelelő” alkalmazás képes megtenni. Mindössze annyi a dolgunk, hogy a sablonunkból valami hasznosat készítsünk.

Csinálj egy könyvtárat valahol a home-odon belül elnevezve azt py_template-nek. Lépj be a könyvtárba és adj hozzá egy alkalmas ikont. Az enyém egy mosolygó arc.

A felület elkészítése

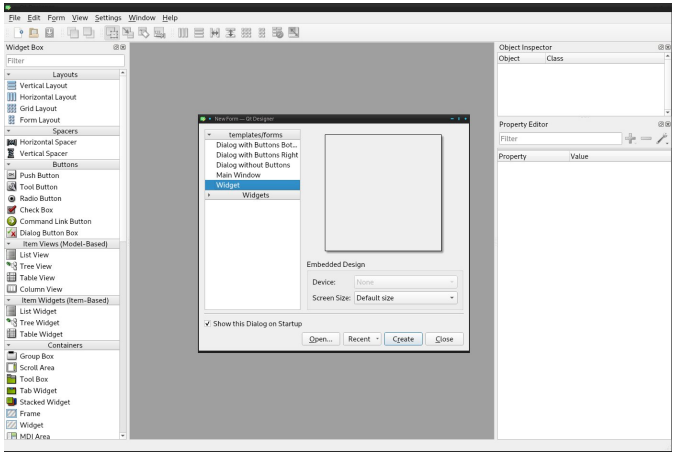
A Qt toolkit (eszköztár) és a dokumentáció komplikált valami – igen! Vess egy pillantást de!

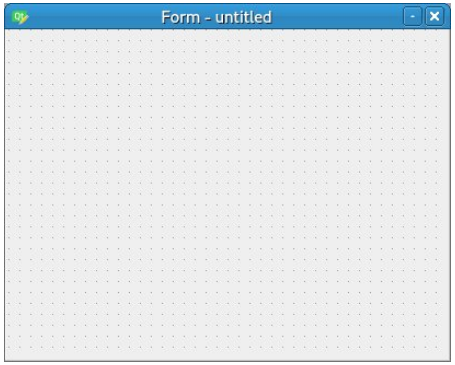
Ez elég arra, hogy még a legkeményebbeket is elriassza. Ugyanakkor a Qt5 szép grafikus eszközzel érkezik, a neve „Qt Designer”, amit már korábban telepítettél és most a menüben ott kell lennie, valahol a fejlesztőeszközöknél, vagy hasonlónál. Ez készíti el a képernyőn látható grafikus felületet. Kell még néhány sor Python kód a felület használata érdekében és van még egy sor apróbb lépés, ami ahhoz kell, hogy összehozzuk a dolgokat.

A következő szakasz úgy tűnhet, mintha túl sokat kellene dolgozni egy egyszerű alkalmazáshoz, de egy sablont készítünk, ami másolható és újra hasznosítható további alkalmazások készítéséhez.

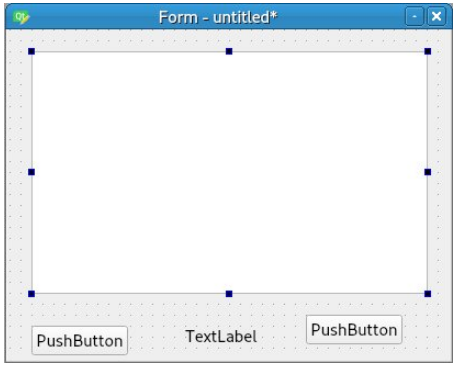
Indítsd el a Designer-t és valami ilyesmit látsz. Az alkalmazott téma és annak függvényében, hogy KDE-t, vagy más környezetet használsz-e, nálad ez egy kicsit másképpen is kinézhet. Esetleg az oldalsáv átméretezésére is szükség lehet, amit az élek vontatásával megtehetsz.

Válassz Widget-et és kattints a Create gombra. Egy új, üres forma jön létre, ami így néz ki (következő oldalon középen):

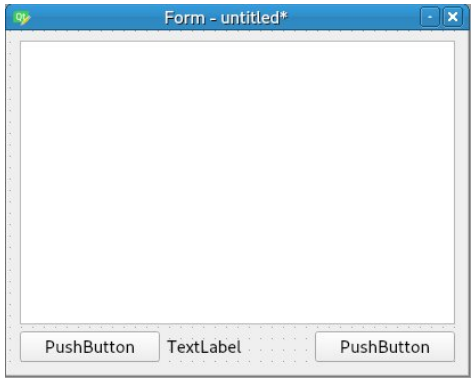




A Widget négyzetében húzz és dobj balra egy textEdit Widget-et az Input widget szekcióból, egy Label-t a Display Widget-ek közül és két Push Button-t a Buttons részből. Helyezd el és méretezd át olyanra, hogy valahogy így nézzen ki.



Jobb gombbal kattints a forma hátterén, a pontozott részen és a felugró menü alján válaszd a Lay Out-ot majd a Lay Out in a Grid-et. Ekkor a Widget-ek egy sokkal szimmetrikusabb formába rendeződnek át, valahogy így:



A tervező alkalmazás jobb oldalánál keresd meg a Property Editor-t. Ha nem lenne ott, keresd meg a View menüben és pipáld ki. Válaszd a left pushbutton-t és a property editorban váltsd az objectName-et clearButton-ra. Görgess le a property editorban a text-hez és váltsd azt „F7 – Clear”-re. Válaszd ki a másik nyomógombot és a tulajdonságait váltsd quitButton-ra és „Esc – Quit”-re. Az objectName-nél a pontos írás fontos, mivel ezt a nevet fogjuk használni a Python kódban. Válaszd ki a fő form-ot a property editor-ban, keresd meg a „window icon”-t és kattints a jobbra tőle lévő üres részre. Nyisd ki a lenyíló mezőt, válaszd a „choose file”-t, menj az új könyvtáradhoz és válaszd ki az oda rakott ikont.

Végül válaszd ki a label-t és változtasd meg a szöveget PyQt5 template-re. Görgess le az Alignement, Horizontal-hoz és kattints a jobb oldali oszlopra. Válaszd ki az AlignHCenter-t. Az űrlapod most úgy nézhet ki mint a fenti befejezett alkalmazás.

Nyomj egy CTRL+R-t és előnézetet mutat neked, amit átméretezhetsz. Vedd észre, hogy a tartalma automatikusan átrendeződik, ahogy változtatsz. A widget-ek még nem csinálnak semmit, mivel némi Python kód kell még hozzájuk. Az előnézetben jobbra fent használd a Close gombot.

Menj a **File** → **save**-hez, navigálj a sablonkönyvtáradhoz és mentsd a fájlt oda template.ui név alatt (az ui kiterjesztés a user interface – felhasználói felület – jelölése).

A könyvtárad most valahogy így nézhet ki:

qt5_template/
face-smiley.png, vagy ahogy az ikont elnevezted
template.ui

A template.ui fájlt a Designer készítette, de a Python számára még nem olvasható. Most ezt fogjuk kijavítani. Készíts egy új fájlt a következő tartalommal:

```
#!/usr/bin/env bash
pyuic5 template.ui > template_ui.py
```

Mentsd ***update_res.sh*** néven. Ez konvertálja a Designer által készített template.ui fájlt a Python által érthető kódra. Legyen az ***update_res.sh***-t futtatható. Dolphin-t használva jobb kattintás a fájlon és a megnyíló párbeszédben a Tulajdonságoknál a Jogosultságok fülnél pipáld ki a „Futtatható” négyzetet.

Ha most futtatod az update_res.sh-t (dupla kattintás) készül másik fájl template_ui.py néven. Ez a Designer által készített fájl Python kódos verziója.

A Python kód

Először be kell állítanunk a geany-t, hogy a Python kódot használja. Nyisd meg a geany-t és menj a

Szerkesztés → Beállítások → Szerkesztő-höz. A Behúzások fülnél a **Width** legyen 4 és a Típus: Szóköz. A Megjelenítés fülnél jelöld be a Sorszámok megjelenítését. Alkalmaz és Ok.

A Python nyelv behúzásokat használ a kód formálása, négy szóköz a szokásos. A most beállítottak azt biztosítják, hogy mindig ilyen legyen, így a tabulátor gomb lenyomásával nem tabulátor karaktert, hanem négy szóközt szúr be, ami majdnem úgy néz ki. Más szerkesztőknél ez másképpen mehet. Ragaszkodj a konvenciókhoz és az életed könnyebb lesz.

Írd be a következő kódot, **pontosan** ahogy látszik – a sorszámok nélkül, de az üres sorokkal együtt. A kód nagybetűérzékeny. A másolás és beillesztés nem javallt, mivel a szedés tartalmazhat láthatatlan formázó karaktereket, amik tönkreteszik a kódot.

Megjegyzés: az összes be nem húzott sor első karaktere az oldal bal szélére fekszik fel, a 10. sor behúzott 4 szóközzel, a 11. további 4-gyel, a 19 ismét 4-gyel. A 10., 11. és 32. sorban megjelenő hosszú vonalak dupla aláhúzások „__”, tehát így kell begépelni. (**Szerkesztői megjegyzés**: a 9., 11., 15. és 15. egy-egy sorban írandó, nem törik meg.)

```
1 #!/usr/bin/env python3
2
3 import sys
4 from PyQt5.QtCore import *
5 from PyQt5.QtWidgets import *
6 import template_ui
7
8
9 class Template(QWidget, template_ui.Ui_Form):
10     def __init__(self):
11     super(self.__class__, self).__init__()
12     self.setupUi(self)
13     self.textEdit.setFocus()
14     self.clearButton.clicked.connect(self.clearText)
15     self.quitButton.clicked.connect(self.exitApplication)
16
17 def keyPressEvent(self, e):
18     if e.key() == Qt.Key_Escape:
19         self.exitApplication()
20     if e.key() == Qt.Key_F7:
21         self.clearText()
22
23 def clearText(self):
24     self.textEdit.clear()
25     self.textEdit.setFocus()
26
27 def exitApplication(self):
28     self.close()
29     sys.exit()
30
31
32 if __name__ == '__main__':
33     app = QApplication(sys.argv)
34     form = Template()
35     form.show()
36     app.exec_()
```

Ments el *qt5_template.py* néven.

Az üres sorokat nem számítva 28 sor. Mielőtt a kódok magyarázatába kezdenék, lássuk, le tudjuk-e futtatni. Ha már fut, akkor megvan a sablon alkalmazásunk.

A könyvtárad most valahogy így néz ki:

```
qt5_template/
   face-smiley.png
   qt5_template.py
   template.ui
   template_ui.py
   update_res.sh
```

Váltsd a qt5_template.py-t futtathatóvá és futtasd. Ha rendben ment, akkor a sablon alkalmazásod megjelenik. Ha igen, akkor gratulálok, készen vagy. Ha nem, akkor ellenőrizd a gépelést – ez a legnehezebb a programozás elején. Többször végig kell menned rajta, mielőtt végre rendbe rakod.

A kód futtatásának melléktermeke, hogy a Python készít egy újabb könyvtárat __pycache__ néven és berak néhány fájlt. Ezzel nem kell és nem szabad foglalkoznunk.

A kód értelmezése

Majdnem a teljes kód a Qt5-re jellemző. Ez Python, de elég fejlett python-kód. Ugyanakkor nem szükséges megérteni, hogy használhassuk. Majd, ahogy jobban megismered, egy része érthetőbbé válik és képes leszel kiegészíteni, módosítani és fejleszteni, hogy a saját alkalmazásoddá váljon. Az elején a kód komplikáltnak tűnhet, különösen, ha még nem nagyon programoztál eddig, tehát ne várd, hogy megértsd. Javaslom, fusd át a magyarázatot és később térj ide vissza, ha már egy kicsit többet foglalkoztál python-nal, qt programozással. Még semmit sem tudunk a Python-ban és Qt-vel történő programozásról és már csináltunk egy teljesen működő grafikus alkalmazást. Ahogy a sokkal hasznosabb alkalmazások felé haladunk, egyre jobban fogjuk látni, ahogy több és több dolog kapcsolódik össze és ezután állhatsz neki a saját alkalmazásaid összerakásának.

Rövid magyarázat:

1. sor: A Python értelmező nyelv és a rendszernek tudnia kell, hogy melyik értelmezőt használja a kód végrehajtásához. Ez a sor mondja meg, hogy python3-at használjon. Ha az update_res.sh fájlt nézed, az első sora hasonlóképpen mondja meg a rendszernek, hogy a bash-értelmezőt kell használni.

Ha ezt a sort nem tennénk bele, akkor a kód futtatására a Python-t a terminálba gépelve vehetnénk rá

python3 qt_template.py

a qt5_template könyvtárunkban állva. A sort tartalmazva ez automatikus lesz.

3-6. sor: fontos utasítások, amikkel a Python számunkra már megírt kódokat „importál”, vagy olvas be.

- A 3. sor a sy modult importálja, amely kódot a Python fejlesztői a rendszerhez való csatlakozás érdekében készítettek nekünk.
- A 4. és 5. sor a Riverbank computing kedves fickói általkészített kódot importálja, amivel a Python megérti a Qt5 rendszert.
- A 6. sor egy olyan kódot hoz, amit a designer-rel a felhasználói felületből generáltunk. Most még nem kell ismernünk ezt a kódot.

9. sor: az alkalmazás alapját jelentő „class”-t (osztály) hoz létre. A class neve Template, nagy „T”-vel, miközben a template_ui kisbetűs „t”. Ez tesz különbséget a class és az importálandó fájl között.

Ez óriási! Ezt a magyarázatot most nyugodtan átugorhatod.

Az objektumorientált programozás megértésének alapja az osztályok (class-ok) megértése. Az osztály (class) egy objektum általános leírásának alapja, akár a természetben hal, madár, vagy kutya osztályok, a kacsa pedig egy madárfaj megkülönböztető jelekkel, de továbbra is állatként. Itt mi létrehozunk egy Template nevű osztályt és az alkalmazásunk ennek az osztálynak egy példánya lesz. Az osztály a QWidget class-on alapul, amivel a Designer-ben indítottunk, de gombokkal, címkékkel és szövegrésszel egészítettük ki az adott formát és ezeket a változtatásokat mentettük template_ui néven, és ezt fordította le egy Python fájllá.

Ez a sor létrehoz egy osztályt, ami alapvetően QWidget alapú és hozzá adja az űrlaphoz fűzött változtatásainkat. A 6. sor importálásának ez volt az oka. Ezt örökléseként ismerjük. Az osztályunk örököl mindent, ami a QWidget-be van kódolva. Ugyancsak megkap mindent, amit a felhasználói fájlunkban változtattunk. Az osztály definíciója meghívja a QWidget és a Form definícióit.

10-15. sor: inicializálás, vagy beállítások.

- A 13. sor adja a fókuszt a textEdit-nek (szerkeszthető szövegrész), így azonnal kezdhetünk oda gépelni, ahogy az alkalmazás elindul.
- A 14. és 15. sor kapcsolja a gombok lenyomásakor generált jelet egy kódhoz, ami valami hasznosat csinál.

17-21. sor: veszi a billentyűk lenyomását és kapcsolja kódokhoz, ahogy a gombokkal is történik.

23-25. sor: törli a szöveget és visszaadja a fókuszt a textEdit objektumnak. A fókuszt a clearButton objektum lenyomásával vette el.

27-29. sor: mindent bezár.

32. sor: a kódunk végrehajtása valójában itt kezdődik. Most csupán fogadd el a sort így, ahogy most látod.

33. sor: létrehoz egy alkalmazás objektumot. Ezt az importált Qt kóddal teszi.

34. sor: a Template osztály „leírását” használja, hogy egy form-nak nevezett form-ot készítsen. Az elnevezés valójában hozzárendelés, ami az általunk készített „form” osztály egy példányát rendeli hozzá. A név nem szükségszerűen form, bármi lehet, de most ennél maradunk.

35. sor: előkészíti a form-ot a képernyőre.

36. sor: futtatja az alkalmazást!

Technikai megjegyzés:

a 33. sor kódja indít mindent. Itt egy QApplication objektum készült, amit én a nevezett alkalmazáshoz rendelek. QApplication minden Qt alkalmazáshoz kell. Ez az objektum néhány olyan létfontosságú információt kap a rendszertől és a sys.argv hívástól, mint a képernyőméret és a kódunk helye. Az információval felépül egy rendszerhez kapcsolt alkalmazás és elindul az esemény huroknak ismert végtelen ciklus (mivel valamilyen eseményt vár, mint pl. gombnyomás, vagy ablakátméretezés). Amikor az esemény bekövetkezik, bekerül a feldogozandó események sorába. Amikor az eseményt feldolgozta, az alkalmazás visszatér a sorhoz, hogy a következő a eseményt feldolgozza. Ez akkor igaz, ha az esemény nem egy, az alkalmazást bezáró kérelem volt. Ezért a Qt alkalmazások eseményvezéreltként ismertek, amik csak akkor használnak rendszererőforrást, amikor esemény van. A 36. sor indítja az alkalmazást az objektum exec_() eljárásával. Vedd észre az aláhúzást a végén. Ez különbözteti meg az eljárást (method) a Python beépített saját exec() eljárásától.

Tanuljunk Python nyelv használatot

A Qt alkalmazás programozásának tanulása két részből áll. Az egyik az aktuális felhasználói felület mechanikája és a speciális kódok, amik a Python-nak kell az azzal való együttműködéshez. Ez a kód a Qt-rendszerhez része, de Python kód. A második rész a szabványos Python kód, ami az alkalmazás funkcióinak végrehajtásához kell. Ennek a kódnak nem kell foglalkoznia a dolgok grafikus oldalával és terminálban, vagy Python értelmezővel is futtatható. Ebben az írásban mutatok egy példát, ami mindkét típus egy-egy részét használja, de külön választottam a szabványos kód magyarázatát, hogy azt a felhasználói felület felépítésével és kódolásával párhuzamosan tanulmányozni lehessen.

A Python elég barátságos az új felhasználókhoz, ugyanakkor nagyon hatékony, ha már megismerted pár jellemzőjét. Kezdjük az alapokkal. Nyiss terminált (igen, parancssori terminált, amit fokozottan próbálunk kerülni. De most nagyon hasznos.) Gépeld be a python parancsot, nyomj Entert és a Python interaktív interpretere elindult. Ide bármilyen „python-os” dolgot beírhatsz és azonnali eredményt kapsz. Ez kitűnő módja dolgokat kipróbálásának anélkül, hogy programot kellene írnod.

python
Python 3.7.1 (default, Oct 22 2018, 10:41:28)
[GCC 8.2.1 20180831] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>

A >>> az intepreter promptja, ami várja, hogy beírj valamit. Amikor ki akarsz lépni az értelmezőből írd be, hogy quit(), vagy nyomj CTRL+d-t. A Python-ban, ha megjegyzést akarsz fűzni kódhoz, használd a # karaktert. Bármir, ami utána van, a Python figyelmen kívül hagyja. A kommentek arra jók, hogy megmagyarázd a nem egyértelmű kódrészleteket. Használj kommenteket, és amikor pár hónap múlva előveszed a kódot örülni fogsz, hogy így tettél.

A Python számos törzs „típusobjektum”-ot használ, később mindet megtárgyaljuk. Ezek közül az első objektumtípus, amit megnéznék, a számok. A Python jó a számokban. Kezdjük az integer-rel (egész szám) és a lebegőpontos számokkal (a tizedesjeles számok), amik floats-ként is ismertek.

A Python nagy képességű számológép és a következő műveleti jelek használhatók:

+ összeadás
- kivonás
* szorzás
/ valódi (lebegőpontos) osztás, ami lebegőpontos értéket ad
// lefelé kerekítő osztás
ugyanakkor, ha a számok bármelyike tizedes, akkor a kerekített érték is tizedes lesz.
% maradékos osztás
** kitevő

Az első három egyértelmű:

```
>>> 7 + 4
11

>>> 2 + -3
-1

>>> 3.7 - 1.5
2.2

>>> 3.14 * 5.726
17.97964
```

Az osztás egy kicsit komplikáltabb, mivel két formája van, a valódi és a kerekítő osztás.

```
>>> 9 / 3 # Valódi osztás
3.0 # mindig tizedes a végeredmény.

>>> 7 / 3
2.3333333333333335 # a lebegőpontos számokban gyakran fordul elő kisebb hiba
# vedd észre az 5-öt a végén. Ennek semmi köze
# a Python-hoz, hanem ahogy a rendszer
# a lebegőpontos számokat tárolja.

>>> 7 // 3 # kerekítő osztás
2 # kettő egészet ad vissza integer-ként

>>> 7.0 // 3
2.0 # a kifejezésben lévő lebegőpontos szám hatására a visszaadott érték is lebegőpontos.

>>> -5 // 2
-3 # lefelé kerekít ... -3 -2 -1 0 1 2 3 ...
```

% (modulus) az osztás maradványát adja vissza (maradékos osztás)

```
>>> 14 % 3
2

>>> 14 % 3.0
2.0

>>> 14 % -3
-1 # Ez nem tűnhet egyértelműnek, de korrekt.
```

** hatványozás

```
>>> 4 ** 2 # 4²
16

>>> 4 ** -2 # 1/4²
0.0625

>>> 4 ** 0.5 # √4
2.0
```

Nagyon hasznos kiegészítők a „kiterjesztett (augmented)” műveleti jelek:

+=, -=, *=, /= és egy csomó további, de ezek a leggyakoribbak.

```
>>> n = 3
>>> n += 2
>>> n
5

>>> n /= 2
>>> n
2.5
```

A következő numerikus függvényeket alkalmazza:

abs(x) az x abszolút értékét adja vissza

divmod(x, y) két egész, az x y-nal való osztásának hányadosát és a maradványt adja vissza

pow(x, y) x-et y-odikra emeli, miképpen azt a ** műveleti jel teszi

pow(x, y, z) Az (x ** y) % z alternatívája

round(x, n) x-et n. egész számra kerekíti
ha az n negatív szám, vagy x-et
n tizedes jelig kerekíti, ha az n pozitív egész

Mindezen alap dolgok mellett komplex számok is elérhetőek valós és képzetes résszel együtt

```
>>> c = 4 + 2j
>>> c
(4+2j)

>>> c.real # valós
4.0

>>> c.imag # képzetes
2.0
```

A számok bináris, hexadecimális, vagy oktális formában is megadhatók, illetve egymásba konvertálhatóak a bin(), hex(), oct() and int() függvényekkel.

```
>>> bin(42)
'0b101010'

>>> hex(42)
'0x2a'

>>> oct(42)
'0o52'

>>>> int(0x2a)
42
```

Vannak még bitenkénti, logikai és összehasonlító operátorok is.

Az operátorok sorrendje a PEDMAS (zárójel, exponenciális, szorzás, osztás, összeadás és kivonás) kiterjesztett szabályt követi balról jobbra végrehajtva.

```
10 + 8 / 2 * 4          >>>26 # nem 11, 36, 56, 36 vagy 2.25.
```

Műveleti jelek:

()	zárójel
**	kitevő
+x, -x, ~x	plusz és mínusz előjel bitenkénti NOT (nem)
*, /, //, %	szorzás, osztás, kerekítő osztás, maradékos osztás (modulus)
+, -	összeadás, kivonás
<<, >>	bitenkénti léptető operátor
&	bitenkénti AND (és)
^	bitenkénti XOR (kizáró vagy)
|	bitenkénti OR (vagy)
==, !=, >, >=, <, <=, azonos, nem azonos, tartalmazz, nem tartalmazza hasonlító, azonosító és tagsági operátorok	
not	logikai NOT (nem)
and	logikai AND (és)
or	logikai OR (vagy)

Gyakran jobb zárójeleket használni, hogy egy adott műveleti sorrendet kikényszeríts.

```
>>> 10 + (8 / 2) * 4
26.0

>>> (10 + 8 / 2) * 4
56.0

>>> (10 + 8) / (2 * 4)
2.25
```

Ha ez mind még nem lenne elég importálhatsz különböző modulokat, vagy csomagokat a képességek bővítéséhez. A matematikai modul mindent tartalmaz, amit egy tudományos számológéptől elvárhatsz, a decimális modul a lebegőpontos számításokhoz állítható pontosságot ad, a törtek modul lehetővé teszi a törtekkel való munkát. Vannak még olyan komoly modulok, mint a numpy, amiket olyan intézmények mint a CERN és a NASA is használ.

Tisztában kell lenned a matematikai alapokkal és kiterjesztett operátorokkal mivel gyakran használtak. A többi várhat addig, amíg szükség nem lesz rájuk, vagy nem találkozol velük példákban.