Gehigarriak

Python-i buruz

Zergatik aukeratu dugu Python? Bere abantaila ugariengatik. Lengoai interpretatua da, sintaxi oso erraza duena, eta ikasteko oso erraza da. Ez zara kezkatu behar (ez asko behintzat) datu motaz. Kontuan hartu behar dugun bakarra zera da: bloke bakoitzeko tabulazioak errespetatzea.

Hala ere, ez da ahaztu behar helburua ez dela programazio lengoai hau ikastea, baizik eta programazioaren funtsak ikastea; eta Pythonek zeregin hori errazten du.

Gainera, oso lengoai erabilgarria da, oso hedatua eta profesionalki erabilia. Hori gutxi balitz, programatzaileek asko estimatzen dute, eta horrek kodea, liburutegi mordoa eta elkar laguntzen duten pertsonen komunitate izugarria dakar.

Pythonek bi bertsio berezi ditu, 2 eta 3. Liburu honetan 3. sintaxia eta estiloa erabili dugu, gaur egungoa delako eta 2. bertsioa desagertutzat ematen delako.

Python zure ordenadorean instalatzen

Web bidez landu badaiteke, baina nahi baduzu zure ordenagailuan instalatu, hemen duzu nola egin. Nahikoa da python gunera joatea eta 3. bertsioaren instalatzailea deskargatzea. Instalazioa pixka bat aldatzen da zure sistemaren arabera, baina, funtsean, honako hau litzateke:

- Windows: instalatzailea deskargatu, exekutatu eta urrats bakoitza berretsi.
- Mac: berdin-berdin.
- Linux: ziur aski berez instalatuta izango duzu, edo, beharbada, ez duzu azalpenik behar 😉

Behin Python instalatuta, programak idatzi eta horiek exekutatu behar dira kontsolan. Kontsolan, gure Python programa dagoen direktorio berdinean horrelako zerbait exekutatuko dugu:

```
python3 programa.py
```

edo

```
python programa.py
```

Kode editoreak

Koderako editore bat erabili nahi baduzu, aukera asko dituzu, baina honako hauek nabarmenduko ditugu:

- [pycharm] (http://www.jetbrains.com/pycharm /) Editore profesionala eta doakoa. Pisutsua.
- [code] (https://vscode.io) Visual Studio Code, Microsoft-ek garatutako editorea. Arina.
- [pydev] (http://pydev.org)
- [sublime] (http://www.sublimetext.com)

Test unitarioak

Test unitarioak kodea ondo eginda dagoela egiaztatzen duten programak dira. Funtsean, funtzioak emaitza zuzena dela egiaztatzen duten programak dira. Testak egiteko hainbat liburutegi daude, baina Python-en `unittest` lehenetsia dago, eta, beraz, ez dago ezer instalatu beharrik.

Demagun kalkulagailu bat errepresentatzen duen klase hau dugula:

```
class Kalkulagailua:
  def batu (self, a, b):
    return a + b

  def kendu (self, a, b):
    return a - b

  def biderkatu (self, a, b):
    return a * b

  def zatitu (self, a, b):
    return a / b
```

Mota horretako test unitarioak egiteko, nahikoa litzateke beste klase hau sortzea, eta hori `unittest` liburutegiaren testa-mota baten oinordekoa izango litzateke. `Kalkulagailua` bera ere inportatu behar dugu, proban jarri behar baitugu. Klaseko funtzio bakoitzak kalkulagailuaren funtzio bakoitza egiaztatzen du. Nahi adina test egin daitezke, funtzioek behar dutena egiten dutela frogatzeko. Ikus dezakezunez, test bakoitzak funtzio bat exekutatzen du funtsean, eta emaitza esperotakoa dela egiaztatzen du, honako hau erabiliz:

```
def test_batu (self):
  kal = kalkulagailua.Kalkulagailua()
  self.assertEqual(kal.batu(40, 2), 42)
```

kalkulatzailea.test.py` fitxategi osoa honako hau izango litzateke:

```
import unittest
import kalkulagailua

class TestStringMethods(unittest.TestCase):
    def test_batu(self):
        kal = kalkulagailua.Kalkulagailua()
        self.assertEqual(kal.batu(40, 2), 42)

    def test_kendu(self):
        kal = kalkulagailua.Kalkulagailua()
        self.assertEqual(kal.kendu(40, 2), 38)

    def test_biderkatu(self):
```

```
        kal = kalkulagailua.Kalkulagailua()
        self.assertEqual(kal.biderkatu(40, 2), 80)

    def test_zatitu(self):
        kal = kalkulagailua.Kalkulagailua()
        self.assertEqual(kal.zatitu(40, 2), 20)

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

Orain nahikoa da testen fitxategia exekutatzea, eta hau ikusiko dugu:

```
python3 kalkulagailua.test.py

....
- ----------------------------------------------------------------------
Ran 4 test in 0.000s

OK
```

Python proiektu bati hasiera

`virtualenv` Python proiektu bat hasteko modu gomendagarri bat izango litzateke: gure sistema edozein dela ere funtziona dezakeen proiektu-karpeta bat sortzeko. Hori dela eta, proiektua malguagoa eta beste ordenagailu batzuk eramateko errazagoa da.

Komando hauek sistemaren kontsolan idatzi beharko dituzu.

```
pip3 install birtualenv
Defaulting to user installation because normal site-packages is not
writeable
Collecting virtualenv
Downloading virtualenv-20.0.31-py2.py3-none-any.whl (4.9 MB)
|. | 4.9 MB 447 kB/s
Collecting distlib < 1, > = 0.3.1
...
```

`virtualenv` proiektu berri bat sor dezakegu honela:

```
virtualenv proiektua
```

Eta horrek `proiektua` izeneko karpeta sortuko du. Jarraian, proiektuaren ingurune birtuala aktibatu behar dugu, honako hau exekutatuz:

```
source proiektua/bin/activate
```

Orain, kodea gehitu dezakegu edo dependentziak instalatu. Depententziak gure proiektuan erabiliko ditugun liburutegien zerrenda da. Horretarako, `requirements.txt` izeneko testu-fitxategi bat sortzea komeni da. Fitxategi horrek formatu hau izan behar du:

```
#Bertsio zehatz bat instalatzeko
#pakete_izena == bertsioa

#Bertsio berdina edo handiagoa instalatzeko
#pakete_izena >= bertsioa

#Bertsio berriena instalatzeko
#pakete_izena
```

Adibidez, `pygame` eta testing pakete bat instalatu nahi baditugu, honako hauek jar ditzakegu:

```
pygame = 1.9.6
unittest
```

Eta gero pakete hori eta beste batzuk instalatu genitzake, `pip3` (edo `pip`) pakete kudeatzailerarekin:

```
source bin/activate
pip3 install -r requirements.txt
```

Beste aukera bat da `pip3`ekin zuzenean liburutegiak instalatzea:

```
pip3 install pygame
```

Instalatuta dagoela egiaztatuko dugu:

```
pip3 list
Package Version
---------- -------
pip 20.2.2
pygame 1.9.6
setuptools 49.6.0
wheel 0.35.1
```

Eta `freeze` erbiliz, pakeeteak `requirements.txt` fitxategian gordeko dugu:

```
pip3 freeze > requirements.txt
```

Orain, pygame erabiliko duen fitxategi bat sor dezakegu:

```
import pygame

pygame.init()

# Set up the drawing window
screen = pygame.display.set_mode([500, 500])

# Run until the user asks to quit
running = True
while running:

    # Did the user click the window close button?
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Fill the background with white
    screen.fill((255, 255, 255))

    # Draw a solid blue circle in the center
    pygame.draw.circle(screen, (0, 0, 255), (250, 250), 75)

    # Flip the display
    pygame.display.flip()

# Done! Time to quit.
pygame.quit()
```

Honela jokatuko genuke:

```
python3 game.py
```

Eta Python-en ingurune birtuala amaitzeko, nahikoa litzateke:

```
deactivate
```

Oharra: `pygame` Python lengoaiarekin bideojokoak garatzeko liburutegia da.