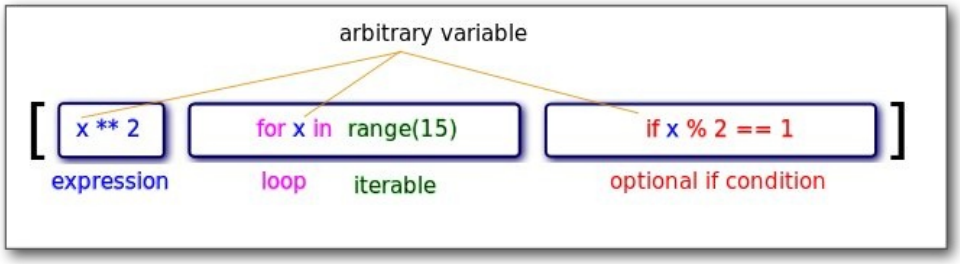
Alkalmi Python - 12. rész

PCLinuxOS Magazine - 2020. janár

Írta: Peter Kelly (critter)

Értelmező

Pythonban értelmező (comprehension) készíthető listához, beállításhoz, vagy szótárhoz. Az értelmező egy bemeneten kapott sorozatból készít egy új



sorozatot az elemekre egy adott kimeneti kifejezést alkalmazva. A sorozatot egy tetszőleges változóval iterálja, ami szintén a kifejezésben szerepel. Opcionális előfeltevés beállítható a kimenetre jutó adatok szűrésére. Zavaros? Nem meglepő, de valóba elég egyszerű. Egy példa talán segít.

Íme egy lista értelmezőkről.

```
>>> single_list =['one', 2, 3, 4] # sima lista
>>> double_list =[i * 2 for i in single_list]        # dupla lista, i egy listából
>>> print(double_list)  # nyomtat dupla lista
['oneone', 4, 6, 8]
```

Az i * 2 kifejezést alkalmazza az eredeti lista valamennyi elemére új listát készítve. A szögletes zárójel mondja meg a Pythonnak, hogy ez egy „lista” értelmező. Az első iterációban az i * 2 kifejezést a „one”-ra alkalmazta, ami a „oneone” sztringet eredményezte. Ezután a 2 integerre alkalmazta a kifejezést, 4-et kapva. Minden eredmény az új dupla listába kerül. A Python a listára ismételve alkalmazza a kifejezést. A kifejezésnek az elemekre alkalmazható kell legyen, ellenkező esetben kivételt dob. A kifejezés alkalmazható az eredeti lista részére is.

Download the Casual Python Source Code

```
>>> double_list =[i * 2 for i in single_list[1:3] ]
>>> print(double_list)
[4, 6]
```

Az iterálandó lista röptében is készülhet és a kifejezés tetszőlegesen összetett lehet.

```
>>> import os, glob
>>> glob. glob('*. py')
['dashboard_ui. py', 'qt_dashboard. py', 'partitions. py',
 'wm_funcs. py', 'get_sensors. py']
>>> tmp_list =['/tmp/'+ i for i in glob. glob('*. py') ]
>>> print(tmp_list)
['/tmp/dashboard_ui. py', '/tmp/qt_dashboard. py',
 '/tmp/partitions. py', '/tmp/wm_funcs. py', '/tmp/get_sensors. Py']
```

Listaértelmezés megfelelő előfeltételt alkalmazva használható a lista szűrésére. Egy példa arra, hogy egy sorozat kizárólag páratlan számai négyzetét kapjuk.

```
result =[i ** 2 for i in range(15) if i % 2]
>>> print(result)
[1, 9, 25, 49, 81, 121, 169]
```

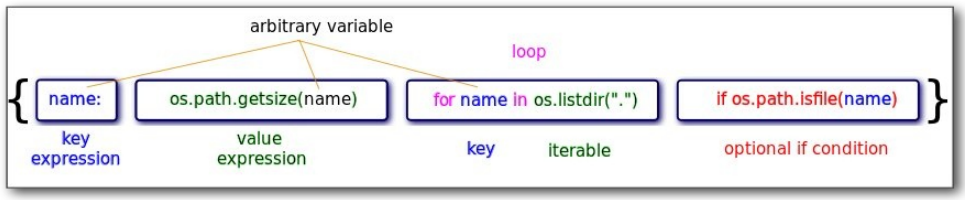
vagy fájlok fájlméret szerinti szűrésére

```
>>> med_files =[i for i in glob. glob('*. py') if 1300 <
 os. stat(i) . st_size < 2000]
>>> print(med_files)
['partitions. py', 'wm_funcs. Py']
```

Szótár értelmezése

Akár a listaértelmezésnél, az eredeti objektum elemeire függvény alkalmazásával egy új adatbázis objektum készül. Ugyanakkor ez esetben szótárobjektum készül, miközben kapcsos zárójelbe foglalt értelmezést alkalmazza.

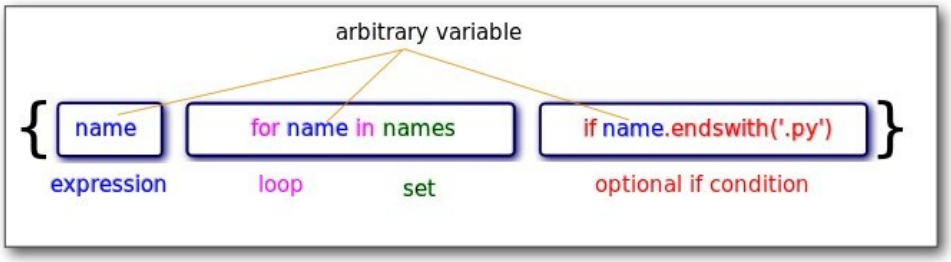
```
>>>  d = { 1: 'eins', 2: 'zwei', 3: 'drei', 4: 'vier', 5: 'fünf'}
>>>   d_upper = {n: d. get(n) . upper() for n in d}
>>>  print(d_upper)
 {1:'EINS', 2: 'ZWEI', 3: 'DREI', 4: 'VIER', 5: 'FÜNF'}
```



Kiválasztás a szótárból.

```
>>> d_upper = {n: d. get(n) . upper() for n in d if n > 2}
>>> print(d_upper)
 {3: 'DREI', 4: 'VIER', 5: 'FÜNF'}
```

Sorozat értelmezése



Listában lehetnek duplikátumok, ám sorozatban nem. Sorozat értelmezése használható a duplikátumok eltávolítására. A eredményezett sorozat visszaalakítható listává a list() típusdefiniálóval.

```
>>> names =['obama', 'Bush', 'clinton', 'bush', 'REAGAN',
 'carter']
>>> cap_names = { name[0] . upper() + name[1: ] . lower() for name in
 names}
>>> print(cap_names)
 {'Bush', 'Reagan', 'Carter', 'Clinton', 'Obama'}
```

Map, filter, reduce és zip

Ez négy, nagyon hasznos, Pythonba beépített függvény.

Download the Casual Python Source Code

Map # térkép

A beépített map() minden elemen egy függvényt alkalmaz ismétlődően minden elemen. Ahelyett, hogy az elemeken lépkednénk, térképet használhatunk. Gyakrabban a lambda függvényt használják. A map függvény map objektumot ad visszsza, amit használhatóvá kell konvertálni. Itt a list() függvényt használjuk,

```
>>> from math import pi
>>> diameters =[1, 2, 3, 4, 5]
>>> areas = list(map(lambda x: x * pi, diameters))
>>> print(areas)
[3. 141592653589793, 6. 283185307179586, 9. 42477796076938,
12. 566370614359172, 15. 707963267948966]
```

Az elemeket át is lehet adni egy függvénylistának.

```
>>> def squared(x) :
…         return x ** 2
…
          >>> def cubed(x) :
…         return x ** 3
…
>>> funcs =[squared, cubed]
>>> values =[3, 7, 64]
>>> for x in values:
…         powers = list(map(lambda f: f(x) ,       funcs))
…         print(powers)
…
[9, 27]
[49, 343]
[4096, 262144]
```

Filter # szűr

A beépített filter egy értékkel tér vissza mindig, amikor egy függvény True-t ad. A visszaadott objektum iteráló, amit a list kap az érvényes elemeket kiválogatására. A célzott lista különféle objektumtípusokat tartalmaz, amik közül csak az integereket vesszük ki.

```
>>> mixed_list =[13, 'songbird', 7, 5, 3.14159, (72, 34) ]
>>> ints = list(filter(lambda x: isinstance(x, int) , mixed_list))
>>> print();ints)
[13, 7, 5]
```

Az utolsó elem, noha integereket tartalmaz, egy sorozat és ezért nem kerül be az eredmények közé.

Reduce # csökkent

Python3-ban a reduce a functiontools modulban van, importálni kell. Valójában ez azt jelenti, hogy többé már nem beépített, de mindig a map-pel, a szűrővel és a zip-pel együtt használatos, így ide helyezték. A reduce függvény végrehajt néhány műveletet (függvényt) az elemeken és visszaadja az eredményt.

```
>>> from functools import reduce
>>> product = reduce((lambda x, y: x * y) ,[2.34, 1.87, 3.92])
>>> print(product)
17. 153136
```

Ez halmozott x*y műveletet végez el a [2.34, 1.87, 3.92] sorozaton x és y változókkal, az eredmény egyetlen érték lesz:17.153136 = 2.34 * 1.87 * 3.92

Zip

A beépített zip függvény nem keverendő össze az elterjedtebb tömörítő eszközzel. A zip generátor, ami egy bejárót ad vissza. Két gyűjteményt vesz és párokba összesíti azokat egészen addig, amíg a rövidebb gyűjtemény el nem fogy.

```
>>> a =[1, 2, 3]
>>> b =[4, 5, 6, 7]
>>> z = zip(a, b)
>>> print(z)
 <zip object at 0x7f62f6a33820>
>>> z = list(zip(a, b))
>>> print(z)
[(1, 4) , (2, 5) , (3, 6) ]
```

For ciklusban alkalmazva párhuzamos iterálás (bejárás) lehetséges.

```
>>> for (x, y) in zip(l1, l2) :
…         print(x, y, '- - ', x * y)
...
2.345 5.678 - - 13.314910000000001
3.456 6.789 - - 23.462784
4.567 7. 89 –- 36.03363
```

Zip akkor hasznos, ha egy kulcs-, és egy értékgyűjteményből készítünk szótárat.

```
>>> partners = dict(zip(
...                         ['Fred', 'Yogi', 'Tom'] ,
...                         ['Wilma', 'Booboo', 'Jerry']))
>>> print(partners)
{'Fred': 'Wilma', 'Yogi': 'Booboo', 'Tom': 'Jerry'}
```

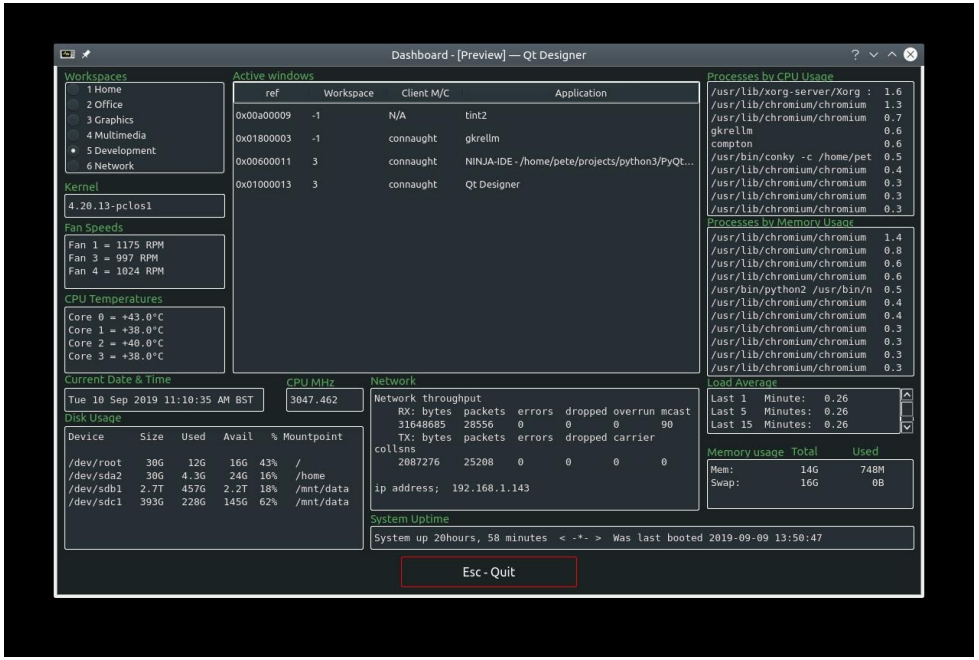
Máskor zip-et szótár feldolgozására használva.

```
>>> partners2 = {k: v for (k, v) in zip(
…         ['Fred', 'Yogi', 'Tom'],
…         ['Wilma', 'Booboo', 'Jerry'])}
>>> print(partners2) {'Fred': 'Wilma', 'Yogi': 'Booboo', 'Tom': 'Jerry'}
```

Dashboard # műszerfal

A mostani bemutató alkalmazásomat dashboard-nak neveztem el. Ez egyetlen ablak, ami számos a Linux-rendszert ellenőrző eszköz végeredményét foglalja magába. Minden egyes eredménycsokor folyamatosan frissül. Ez a változat 14 eredménycsoportot tartalmaz, de esetedben te döntesz, milyen sok, vagy kevés legyen.

Az alkalmazás elkészítéséhez a saját szokásos moduljaim közül kettőt használtam fel, amiket bármikor behívhatok, amikor a működéshez kell. Igazából nem használom fel a modulok összes eljárását. Bemutatom majd a modulokat és a fő alkalmazásszerkezetét, de nem a szokásos részletességgel. Mostanra már értened kell az itt leírt kódot. Kedved szerint sok, vagy kevés eljárást alkalmazhatsz, a számodra hasznosnak tekinthető rendszerfigyelőt építhetsz fel. Ehhez a munkához gondoskodnod kell arról, hogy az összes meghívásra kerülő rendszerparancs, mint a wmctrl, ténylegesen telepítve legyen a rendszeredre.



Ahogy már jeleztem, az alkalmazás a kedvencemet a „dark” témát használja, ám ha te mást szeretsz, akkor a QtDesigner-ben add hozzá, vagy változtasd meg a stíluslapot. Mivel az alkalmazás kicsit erőforrásigényes lehet az adatgyűjtéssel, ezért írtam egy bash-szkriptet a futtatás ki-, és bekapcsolására. Az alkalmazás és a két modul teljes kódja a Magazin honlapjáról letölthető.

A fenti képernyőképen egy Quit pushButton (kilépés) gomb van; az összes elem felcímkézett, a nagy „Active window” megjelenítő egy Qtable widget; a munkaterületeket („Workspaces”) hat QradioButtons jeleníti meg, minden elnevezett munkaterületet egy. Egyszerre csak egy rádiógomb aktiválhat, ami megjeleníti az aktuális munkaterületet. A zöld szövegelemek a címkék és a többi megjelenítő widget QplainTextEdit típusú. A megjelenítő widget-ek formáját hozzájuk stíluslapok határozzák meg.

background- color: #263034;
color: #EDE4E4;
border: 1px solid white;
border- radius: 2px;

A kialakítás többi része kötetlen. Használd a képzeletedet és formáld kedvedre a dolgokat.

Ez a fő alkalmazás kódszerkezete.

```
  1  #!/usr/bin/env python
  2  # -*- coding: utf-8 -*-
  3
  4
  5  import sys
  6  import os
  7  import subprocess
  8  import socket
  9  import time
 10  from PyQt5.QtCore import *
 11  from PyQt5.QtGui import *
 12  from PyQt5.QtWidgets import *
 13
 14  from wm funcs import getCurrentWorkspace, getManagedWindows
 15  from get sensors import getTemps, getFanSpeeds
 16
 17  import dashboard ui
 18
 19
 20  class Dashboard(QMainWindow, dashboard ui.Ui MainWindow):
181
182
183  if __name__ == '__main__':
184
185      app = QApplication(sys.argv)
186      form = Dashboard()
187      form.show()
188      app.exec ()
189      sys.exit(0)
190
```

5-9. sor a szükséges importálások a szabványos Python-könyvtárból.

10.12. sor importálja a PyQt keretrendszer felhasználandó részeit.

14-15. sor importál néhány függvényt a saját moduljaimból.

17. sor a QtDesigner-ben megformált felhasználói felületet importálja.

Ezután meghatározzuk a Dashboard nevű osztályt, végül létrehozzuk és futtatjuk az alkalmazást.

A kódmagyarázat következő részében átnézzük a Dashboard osztály tartalmát.

```
 20  class Dashboard(QMainWindow, dashboard ui.Ui MainWindow):
 21
 22      def __init__(self):
 23          super(self.__class__, self).__init__()
 24          self.setupUi(self)
 25          self.getKernel()
 26          self.ip = self.getIpAddress()
 27          self.btn_Quit.clicked.connect(self.exitApplication)
 28          QTimer.singleShot(0, self.start_loop)
 29
 30      def keyPressEvent(self, e):
 33
 34      def closeEvent(self, event):
 36
 37      def start_loop(self):
 54
 55      def getDate(self):
 61
 62      def getUptime(self):
 67
 68      def getCpuFreq(self):
 73
 74      def getCpuProcs(self):
 79
 80      def getMemProcs(self):
 85
 86      def getMem(self):
 90
 91      def getDiskSpace(self):
101
102      def getNet(self):
111
112      def getKernel(self):
115
116      def getWorkpaces(self):
126
127      def getWindows(self):
149
150      def getFanRPMs(self):
155
156      def getCoreTemps(self):
161
162      def getLoadAverage(self):
171
172      def getIpAddress(self):
176
177      def exitApplication(self):
181
```

Az osztály 20 eljárást tartalmaz, egy kivételével mind az űrlap különböző widget-jei által létrehozott szöveghez beírandó értékeket adja meg, a kivétel pedig a __init__ eljárás, ami beállít mindent az alkalmazáshoz és a megjelenítőhöz.

```
def __init__(self) :        # initialize the window
        super(self__class__, self) . __init__()
        selfsetupUi(self)
        selfgetKernel()
        # called here as will not change this session
        selfip = selfgetIpAddress()
        # also unlikely to change
        selfbtn_Quit. clicked. connect(selfexitApplication)
        QTimer. singleShot(0, selfstart_loop) # set update timer
```

Egy furcsaság, hogy két eljárást, a getKernel-t és a getIpaddress-t azonnal meghívjuk, amik a teljes futás alatt csak egyszer jelennek meg. Ezen eljárások által jelentett források az alkalmazás teljes futása alatt nem valószínű, hogy változnak, ezért meghívhatók most és bízhatunk a változatlanul maradásukban. Emellett beállítja az alkalmazást, ellenőrzi a Quit gombot, hogy aktiválták-e, ami miatt az alkalmazás leállna. Végül a start_loop() eljárással elindul az időzítő, hogy ellenőrizze a többi eljárás által jelentett változásokat.

A keyPressEvent és a closeEvent eljárások mostanra már megszokottak a kódunkban.

```
        def keyPressEvent(self, e) :     # escape key press
                if e. key() == Qt. Key_Escape:
                        selfexitApplication()
        def closeEvent(self, event) :    # close button click
                selfExitApplication()
```

A start_loop eljárást mindig meghívja, amikor az időzítő „kilő”. A dolga az, hogy minden „kilövés” után várjon egy másodpercet, amíg a dolgok stabilizálódnak, majd feldolgozza az egyes adatgyűjtő folyamatok adatait, frissíti a kijelző tartalmat és a qApp.processEvents() eljáráson keresztül ellenőrzi, hogy az exitApplication eljárást végre kell-e hajtani.

```
        def start_loop(self) :
                # loop through methods when timer fires
                        while True:
                                time. Sleep(1)   # seconds between updates
                                selfgetUptime()
                                selfgetDate()
                                selfgetCpuFreq()
                                selfgetCpuProcs()
                                selfgetMemProcs()
                                selfgetMem()
                                selfgetNet()
                        selfgetWorkpaces()
                                selfgetWindows()
                                selfgetFanRPMs()
                                selfgetCoreTemps()
                                selfgetDiskSpace()
                                selfgetLoadAverage()
                                qApp. ProcessEvents()
```

Az egyes eljárások és rövid magyaráztuk a következő.

getDate()

Veszi a rendszerdátumot és elhelyezi a label_now setText eljárásban. A Python3 UTF-8 kódolást alkalmaz alapból, ezért át kell alakítani, hogy a területi beállításaid mellett értelmezhető legyen. Fogja az aktuális szöveget és a megjelenítésre átadás előtt átformálja.

```
        def getDate(self) :
                now = subprocess. check_output(" date") . decode('utf- 8')
                time_details = now. split()
                now_date = (time_details[0: 4])
                now_time = (time_details[4: ])
                selflabel_now. setText(now)
```

getUptime()

Az uptime parancs -p opciója egy szépen (pretty) formázott időfeliratot képez, az -s opció (since) pedig mutatja, hogy a rendszer mikor boot-olt be. Mindkét kimenetet megformálja a label_uptime.setText() eljárásnak átadás előtt.

```
        def getUptime(self) :
                up = subprocess. check_output(" uptime - p" ,
 shell=True) . decode('utf- 8')
                bt = subprocess. check_output(" uptime - s" ,
 shell=True) . decode('utf- 8')
                up_text = 'System '+ up. Rstrip() + '<- *- >        Was
 last booted '+ bt                selflabel_uptime. setText(up_text)
```

A következő öt eljárás mostanra már nem igényel magyarázatot.

GetCpuFreq()

Kiolvassa a CPU frekvenciáját.

```
        def getCpuFreq(self) :
                f = subprocess. check_output(" lscpu | grep 'CPU MHz'" ,
                        shell=True) . decode('utf- 8')
                freq = f. split()[2]
                selflabel_freq. setText(freq)
```

getCpuProcs()

Kiolvassa a processz-ek listáját a cpu-használat szerint csökkenő sorrendben.

```
        def getCpuProcs(self) :          # first 10 only to fit in display
                cp = subprocess. check_output(
                        " ps - eo cmd, %cpu - - sort=- %cpu | sed '/CMD/d'|
                                head - n10" ,
                        shell=True) . decode('utf- 8')
                        selfplainTextEdit_proc_cpu. setPlainText(cp)
```

getMemProcs()

Kiolvassa a processz-ek listáját a memórihasználat szerint csökkenő sorrendben.

```
def getMemProcs(self) :          # first 10 only to fit in display
        cp = subprocess. check_output(
                " ps - eo cmd, %mem - - sort=- %mem | sed '/CMD/d'| head -
n10" ,
                shell=True) . decode('utf- 8')
        selfplainTextEdit_proc_mem. setPlainText(cp)
```

getMem()

Kiolvassa az aktuális memóriahasználatot.

```
        def getMem(self) :
                mem = subprocess. check_output('free - m | sed - n " 2, 3p" ',
                        shell=True) . decode('utf- 8')
        selfplainTextEdit_mem. setPlainText(mem)
```

Ha nem érted a parancsokat, akkor próbáld meg egymás után végrehajtani azokat terminálban, hogy lásd, mit csinálnak az egyes részek. Például:

```
ps - eo cmd, %cpu - - sort=- %cpu
 ps - eo cmd, %cpu - - sort=- %cpu | sed '/CMD/d'
 ps - eo cmd, %cpu - - sort=- %cpu | sed '/CMD/d'| head - n10
```

A parancs kiadja a kernel által ismert aktuális folyamatokat.

-e = összes processz
 -o cmd, %cpu = megmutatja a folyamatot indító parancsot, amit %-os cpu-használat követ
 --sort=-%cpu = csökkenő sorrendben rendezés (-%) cpu
sed '/CMD/d' = törli az összes „CMD"- tartalmazó sort, így eltávolítja a fejléc sorát
head -n10 = a kiment első 10 sorát jeleníti csak meg.

getDiskSpace()

Kiolvassa a partíciók aktuális használtsági állapotát.

```
def getDiskSpace(self) :
        header = ('Device Size Used Avail % Mountpoint\n\n')
        df_str = ('df - h '+ '- -
                        output=source, size, used, avail, pcent, target'+
                        '| grep /dev/ | sed " s/ / /" | sed - n - e
                        " /^\/dev/p" ')
                # the sed statement replaces 5 spaces with one
                ds = subprocess. check_output(df_str,
 shell=True) . decode('utf- 8')
                ds_text = header + ds
                selfplainTextEdit_disk_space. setPlainText(ds_text)
```

getNet()

Az eljárás egyszerűen meghívja az ip parancsot és a kimenet számára átformálja. Egy dolgot kell itt észrevenni, hogy a hálózati csatoló itt eth0, ami lehet teljesen más is, különösen, ha vezeték nélküli kapcsolatod van. Például, a laptopomon, amin Arch Linux fut, enp6s0 és wlp3s0 az ethernet és a vezeték nálküli kapcsolat azonosítója. Egyszerűen helyettesítsd be a megfelelő nevet a te összeállításodhoz.

```
        def getNet(self) :
                selfplainTextEdit_net. clear()
                throughput = subprocess. check_output(
                        'ip - s link show eth0 | sed - n " 3, 6p" ',
                        shell=True) . decode('utf- 8')
                throughput_text = 'Network throughput\n'+ throughput
                selfplainTextEdit_net. setPlainText(throughput_text)
                ip_str = 'ip addresss: '+ selfip
                selfplainTextEdit_net. appendPlainText(ip_str)
```

getKernel()

Ez elég egyértelmű kell legyen.

```
def getKernel(self) :
        k = subprocess. check_output(" uname - r" ,
                shell=True) . decode('utf- 8')
selflabel_kernel. setText(k)
```

A következő négy eljárás az én moduljaimból hív meg függvényeket.

GetWorkspaces()

A getCurrentWorkspace kimenete elemleíró két sztringgel: az aktuális munkaterület száma és a neve. Ezt a cw változó tárolja. Készít egy listát a hat rádiógomb objektumból (listában bármilyen objektum lehet). A lista elemére a hivatkozás a visszaadott (integer) munkaterületszám alapján történik, aminél a setChecked() eljárás „True”. Ez automatikusan átállítja a többi RadioButtons-t „False”-ra.

```
def getWorkpaces(self) :
        # only interested in current ws number - cw[0]
        cw = getCurrentWorkspace()
        wspace =[selfrb_ws1,
                selfrb_ws2,
                selfrb_ws3,
                selfrb_ws4,
                selfrb_ws5,
                selfrb_ws6, ]
        wspace[int(cw[0]) ] . setChecked(True)
```

getWindows()

Most használjuk először a QtableWidget-et, ezért némi magyarázat szükséges.

A getManagedWindows() függvény egy sztringekből álló listát ad vissza, ami az ablakkezelőd által pillanatnyilag kezelt ablakok információit tartalmazza. A wmctrl egy elég összetett parancs, amit itt nem kívánok részleteiben ismertetni. Beállítjuk a tábla widget sorszámlálóját 0-ra, töröljük a táblázat tartalmát, majd végigléptetünk a visszaadott sztringek listáján.

```
If not window:
        continue
```

Kilép a ciklusból, amikor a lista kifutott.

Minden egyes sztring egy ablakelem-listára tördelt.

Az első oszlopot c0, az első elem tölti ki. Ez az ablakkezelő hivatkozási száma az ablakra. A következő elem, ami a c1-es oszlopba kerül, a munkaterület száma. Amennyiben az érték -1, akkor az ablakra a rögzítés attribútum beállított – minden ablakban meg fog jelenni. Ellenőrizzük a -1 értéket, és ha előfordul, akkor „sticky”-re (rögzített) cseréljük. Ezt nem minden ablakkezelő fogadja el, ahogy a KDE is egy nagy számot ad. Rád bízom a döntést, hogy fontosnak találod-e annyira, hogy átírd azt a részt. A következő elem a c2-be megy és a maradék pedig c3-ba. Készítettünk egy c0, c1, c2 és c3 elemű listát. Végiglépegetve a listán, kitöltjük az egyes oszlopokat, majd növeljük a sorszámlálót és végigmegyünk a következő listaelemeken.

```
def getWindows(self) :
        managed_windows = getManagedWindows()
        row = 0
        selfwindow_table. clearContents()
        for window in managed_windows:
                if not window:
                        continue
                window_items = window. split()
                c0 = window_items[0]
                c1 = window_items[1]
                if c1 == '- 1':
                        c1 = 'sticky'
                c2 = window_items[2]
                c3 = " " . j oin(window_items[3: ])
                col_list =[c0, c1, c2, c3]
                for column in range(4) :
                        selfwindow_table. setItem(row,
                                column, QTableWidgetItem(col_list
                                [column]))
                row += 1
```

getFanRPMs()

Ez a get_sensors modulomból egyszerűen meghívja a getFanSpeeds() függvényt és az egyes kimeneti értékeket sima plainTextEdit widgetnek adja át,

```
def getFanRPMs(self) :
        selfplainTextEdit_fan_speeds. clear()
        fs = getFanSpeeds()
        for fan in fs:
                selfplainTextEdit_fan_speeds. appendPlainText(fan)
```

getCoreTemps

Az előzőeljáráshoz hasonlóan ez is meghív egy függvényt az importált modulból és kiírja a cél widget-et.

```
        def getCoreTemps(self) :
                selfplainTextEdit_core_temps. clear()
                ctmp = getTemps()
                for core in ctmp:
selfplainTextEdit_core_temps. appendPlainText(core)
```

getLoadAverage()

Ez az eljárás az uptime parancsot futtatja, vágja és átformálja a kimenetet, majd az eredményt a plainTextEdit widget-be tölti.

```
        def getLoadAverage(self):
                self.plainTextEdit_load_average.clear()
                up = subprocess.check_output("uptime", shell=True).decode('utf-8')
                averages = up.split('average:')[1].split(', ')
                la1 = 'Last 1 Minute: '+ averages[0]
                la5 = 'Last 5 Minutes: '+ averages[1]
                la15 = 'Last 15 Minutes: '+ averages[2][:4]
                load_average = la1 + '\n'+ la5 + '\n'+ la15
                self.plainTextEdit_load_average.setPlainText(load_average)
```

getIpAddress()

Nem igazán értem ennek a kódnak a működését, de működik, így használom.

```
        def getIpAddress(self):
                s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                s.connect(("8.8.8.8", 80))
                return s.getsockname()[0]
```

exitApplication()

Ezt az eljárást egyenesen a legelső sablonalkalmazásunkból másoltam ki.

```
        def exitApplication(self) :
                '"quit the application"'
                selfclose()
                sys. exit()
```

Amikor a kódot ezen a fájlon futtatjuk, akkor a következő hajtódik végre és reményeink szerint megjeleníti az alkalmazásunkat.

```
if __name__ == '__main__':
        app = QApplication(sys. argv)
        form = Dashboard()
        form. show()
        app. exec_()
        sys. exit(0)
```

A modulok

Az első a saját készítésű moduljaink közül a wm_funcs.py. A kód elég egyszerűen érthető kell legyen. Futtatható szkript-be írtam, ami végrehajtja az összes függvényt és a kiadja az eredményeket. Ez biztosítja a függvények megfelelő működését.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-


import subprocess


def getCurrentWorkspace():
    '''retrieves a list of available workspaces and identifies the
       current one returns a tuple of strings'''

    # get workspace list
    wk spaces = subprocess.check output('wmctrl -d',
                                        shell=True).decode('utf-8')
    # get current workspace
    for line in wk spaces.split('\n'):
        if line:
            status = line.split()[1]
            ws num = line.split()[0]
            ws name = line.split()[8]
        if status == '*':
            return ws num, ws name
        ws current = ws num + ' (' + ws name + ')'


def getWmInfo():
    '''get information about the window manager and the environment
       returns a string'''

    wm info = subprocess.check output('wmctrl -m',
                                      shell=True).decode('utf-8')
    return wm info


def getManagedWindows():
    '''get a list of windows managed by the window manager
       returns a list of strings'''

    wm windows = subprocess.check output('wmctrl -l',
                                         shell=True).decode('utf-8')
    return(wm windows.split('\n'))


if __name__ == '__main__':
    print(getCurrentWorkspace(), '\n')
    print(getWmInfo(), '\n')
    print(getManagedWindows())
```

A get_sensors.py modul:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-


import os
import subprocess

def getTemps():
    ''' returns a list of strings containing
        core information for each CPU core '''

    temps = subprocess.check_output(
                            'sensors coretemp-isa-0000',
                             shell=True).decode('utf-8')
    lines = temps.split('\n')[3:7]
    core_num = 0
    core_info = []
    for core in lines:
        core_info.append('Core ' +
                         str(core_num) +
                         ' = ' + core.split()[2])
        core_num += 1
    return core_info

def getFanSpeeds():
    ''' returns a list of strings containing
        fan information for all running fans '''

    fan_speeds = subprocess.check_output(
                            'sensors it8728-isa-0290',
                             shell=True).decode('utf-8')
    lines = fan_speeds.split('\n')[11:15]
    fan_num = 0
    f_info = []
    for fan in lines:
        rpm = fan.split()[1]
        if rpm == '0':  # ignore non-running fans
            fan_num += 1
        else:
            fan_num += 1
            f_info.append('Fan ' +
                          str(fan_num) +
                          ' = ' + rpm + ' RPM')
    return(f_info)


if __name__ == '__main__':
    print(getTemps(), '\n')
    print(getFanSpeeds())
```

Futtatnod kell a sensors parancsot terminálban, hogy megtudd a rendszeredben lévő szenzorok pontos nevét, majd annak megfelelően átírni a 13. és 30. sorok kódját.

Vezérlőpult kapcsolása

Ez a shell-szkript alkalmas a vezérlőpult alkalmazás ki-, és bekapcsolására. Ha erről a szkriptről indult, akkor a szkript ismétlése leállítja. Ha más módon indult el, akkor nem biztos, hogy elkapja. Legjobb ezt a szkriptet a már megismert indító alkalmazások egyikéből futtatni. Írd át az útvonalat az alkalmazáshoz a rendszerednek megfelelően.

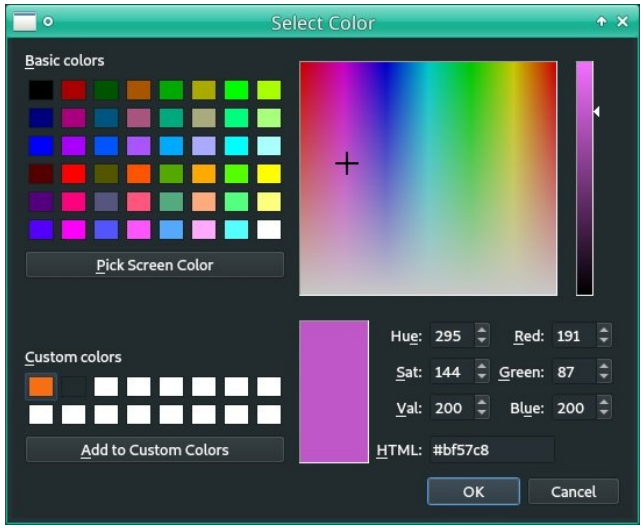
```
#! /bin/bash

# is dashboard already running?
G_STATUS=$(ps aux | grep[q] t_dashboard. py$)
# get the pid if so
G_PID=$(echo $G_STATUS | cut - d" " - f2)
# if there is a pid then is running
if[[$G_PID ! = " " ] ]
then
        # so kill it        kill $G_PID
else
        # otherwise start it
        ~/mag/python/apps/qt5_dashboard/qt_dashboard. py &
fi
# done!
Exit 0
```

A továbbiak

Ezzel eljuottunk olyan messzire, ami még „alkalmi Python”-nak nevezhető. Ettől kezdve minden további már egy sokkal magasabb szintet jelent. Ha követted eddig, akkor elég jó helyzetben vagy ahhoz, hogy megtedd a következő lépést. Ha nem akarnál továbblépni, akkor most már van elég ismereted ahhoz, hogy elkezdj saját alkalmazásokat írni.

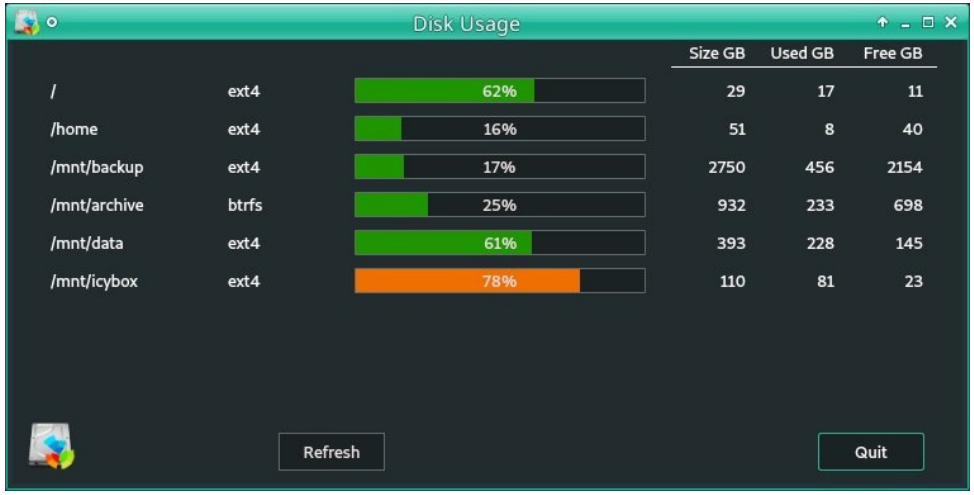
Noha ezek a cikkek grafikus alkalmazások készítésére koncentráltak, nem kell leragadni ennél a Python alkalmazásában. Ahogy az láttuk, a Python rengeteg modullal rendelkezik ahhoz, hogy a rendszerrel együttműködjön és az így nyert adatok szűrhetőek és átalakíthatóak parancssori terminálban megjelenítéshez.

Most gyakorolnod kellene, hogy sokkal könnyebben menjen. Hogy ötletet adjak, mutatok néhány képernyőképet az általam készített alkalmazásokról. Némelyik nagyon egyszerű, némelyik nem ad többet, mint a már megismertek, amíg másokhoz némi kutatásra van szükség, hogy elérhető legyen a funkció.



Szín pipetta

Kikapcsoló alkalmazás



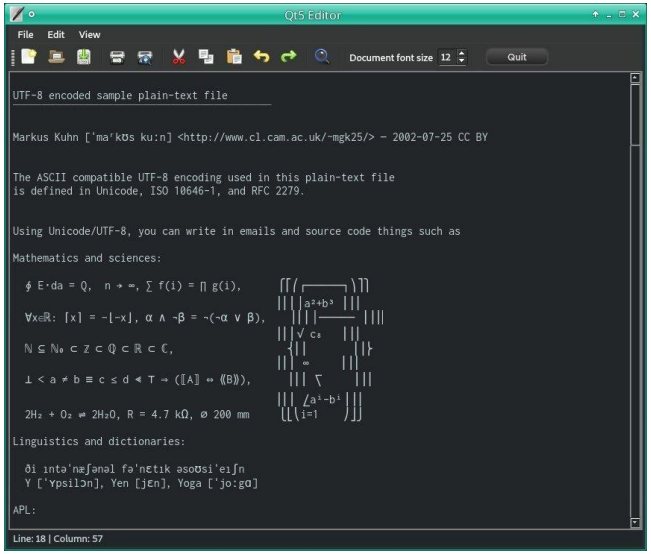
Lemez használtság kijelző

Egyszerű számológép

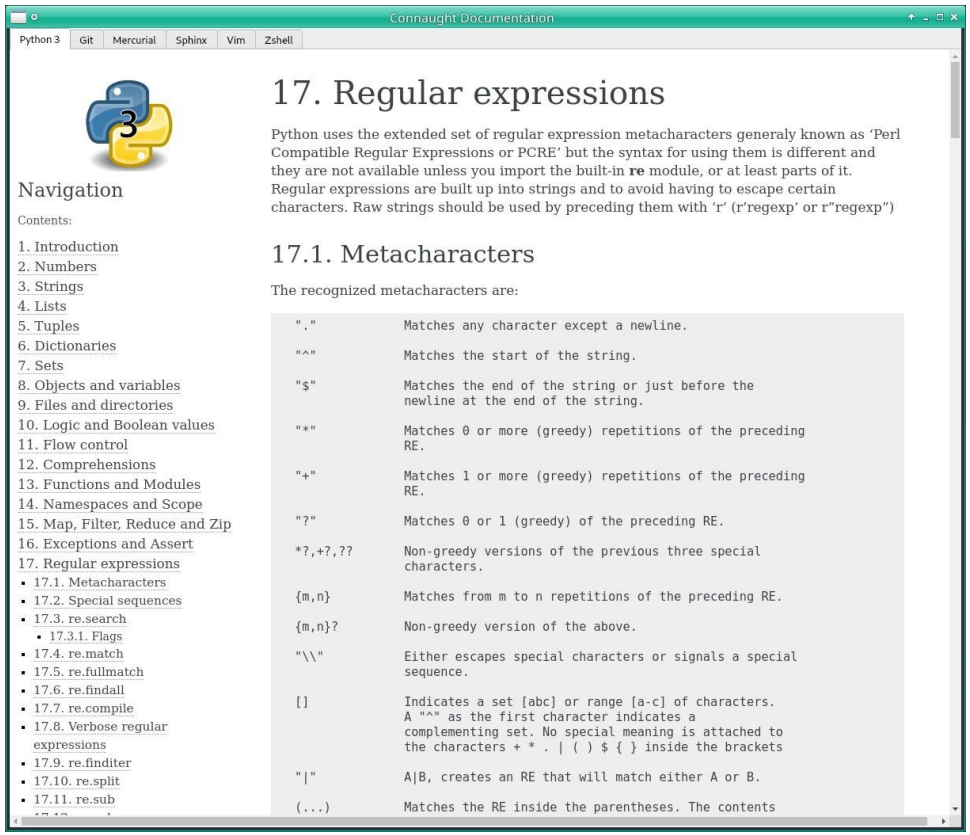
Partíciók csatoló

Munkaterület-váltó

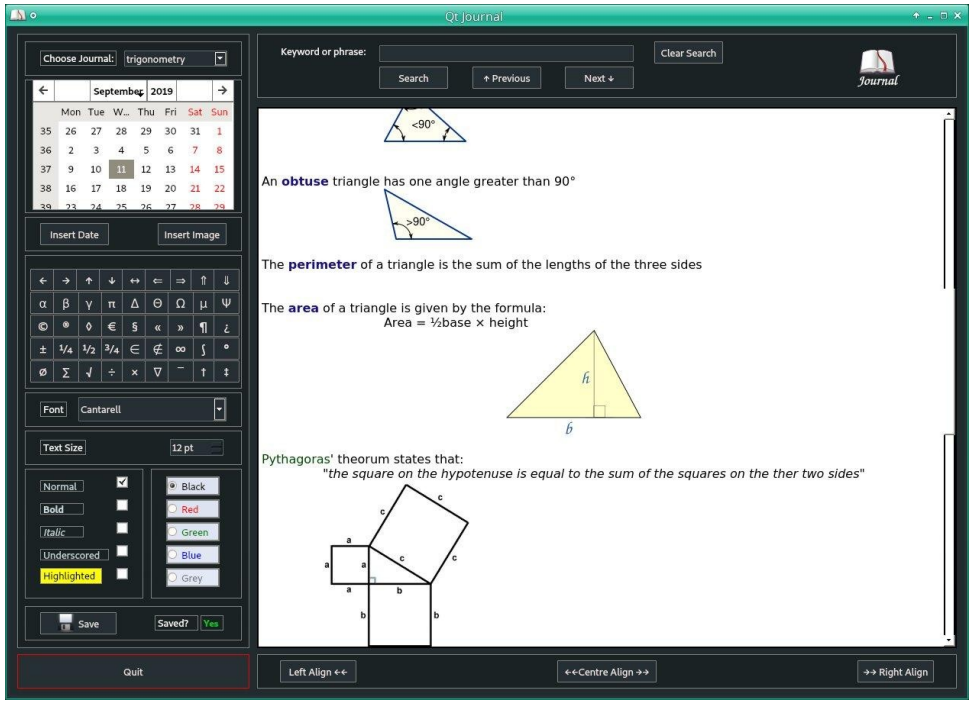
Hangerő szabályozó



Egyszerű szövegszerkesztő

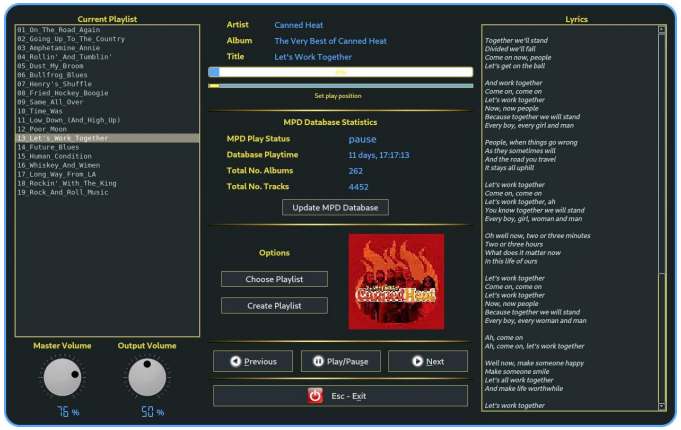


HTML böngésző – a html kódot a sphinx és a Python generálja Dokumentációkészítő

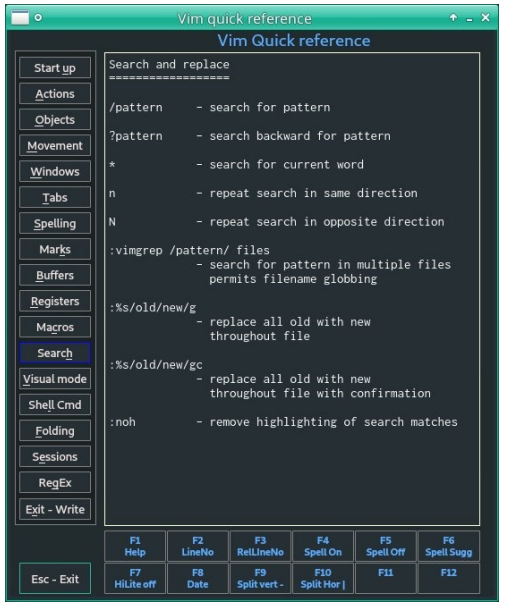


Személyes napló / Feljegyzéskészítő

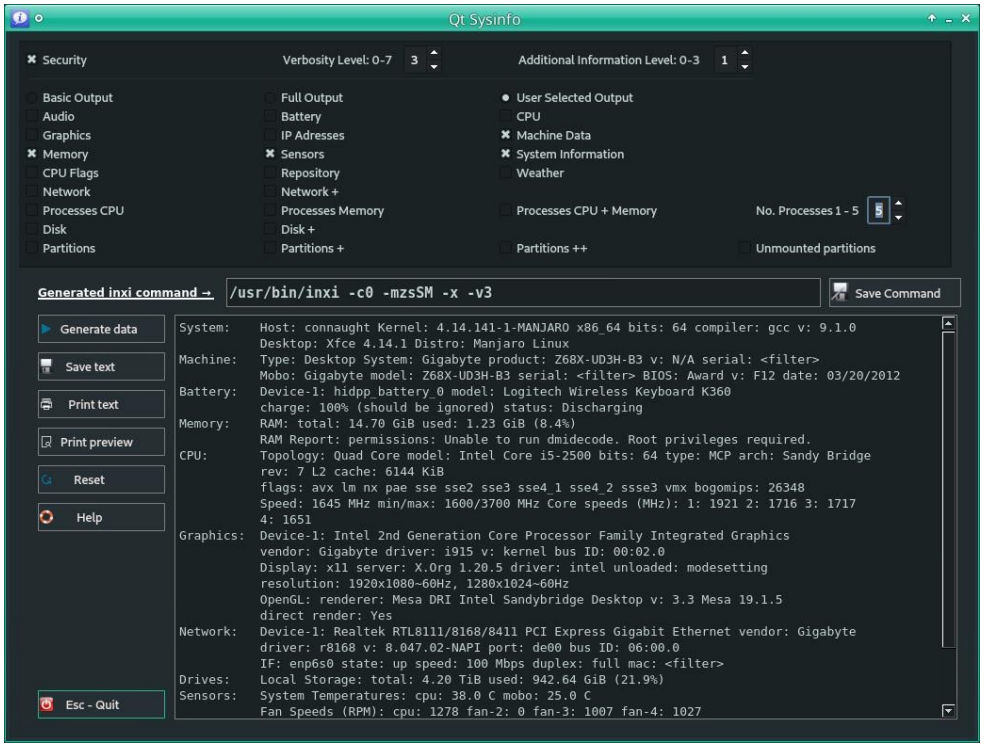
Videólejátszó



Zenelejátszó online dalszöveg letöltővel

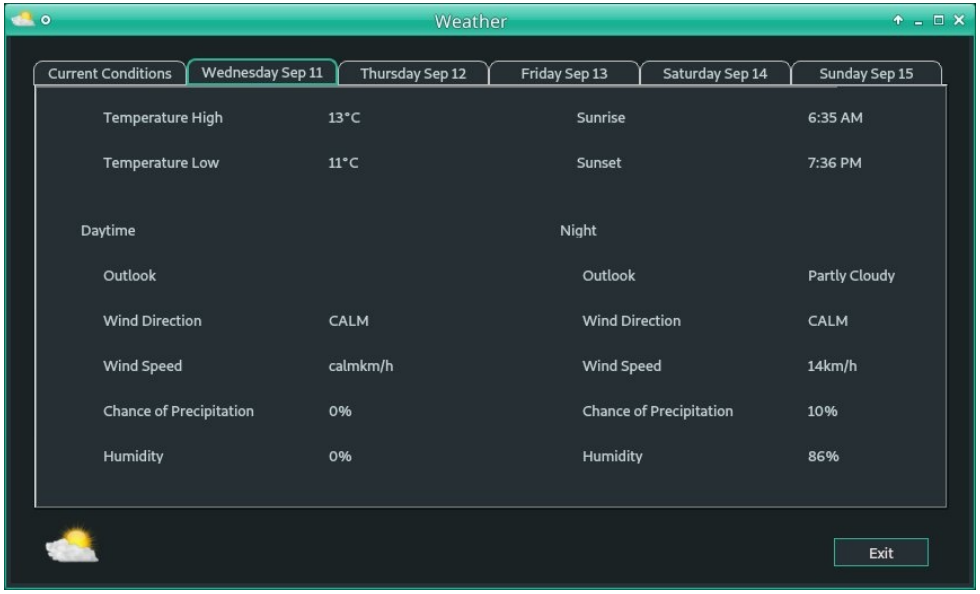


Vim gyors referencia



Inxi rendszerelemző alkalmazáshoz felület

Gyorsbillentyű emlékeztető – billentyűérzékeny



Időjárás-előrejelző