Python 3: programiranje i GUI

za inženjere elektronike



Andrew Pratt

Agencija Eho

www.infoelektronika.net

Sva prava zadržana. Nijedan deo ove knjige ne sme biti reprodukovan u bilo kom materijalnom obliku, uključujući fotokopiranje ili slučajno ili nenamerno smeštanje na bilo koji elektronski medijum sa ili uz pomoć bilo kog elektronskog sredstva, bez pismenog odobrenja nosioca autorskih prava osim u skladu sa odredbama zakona o autorskim pravima, dizajnu i patentima iz 1988. godine ili pod uslovima izdatim od Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London, England W1P 9HE. Prijave za pismene dozvole radi štampanja bilo kog dela ove publikacije upućuje se izdavaču ove knjige. Izdavač je uložio najveće napore da bi se obezbedila tačnost informacije sadržanih u ovoj knjizi. Izdavač ne može da pretpostavi neprijatnosti i ovom izjavom isključuje bilo kakvu odgovornost za bilo koju stranku koja bi imala gubitke ili štetu uzrokovanu greškama ili propustima u ovoj knjizi, bez obzira da li su greške ili propusti nastali usled nemara, nezgode ili bilo kog drugog razloga.

ISBN 978-86-80134-12-3

Naslov originala: Python 3: Programing and GUIs

Izdavač originala: Elektor International Media B.V.

Autor: Andrew Pratt

Prevod: Biljana Tešić

Izdavač i štampa: Agencija Eho

e-mail: redakcija@infoelektronika.net

Sadržaj

POGLAVLJE 1 • LINUX 12
Predgovor drugog izdanja 11
Poglavlje 1 • Linux 13
1.1 • Uvod 13
1.2 • Pribavljanje operativnog sistema 13
1.3 • Instrukcije 13
1.4 • Upravljanje datotekama i direktorijumima 14
1.5 • Džoker znakovi 17
1.6 • Preusmeravanje 18
1.7 • Rad sa Python programima 18
1.8 • Pravo pristupa datotekama 18
1.9 • Komandni skriptovi 19
1.10 • X server 20
POGLAVLJE 2 • POČETAK RADA U JEZIKU PYTHON 21
2.1 • Uvod 21
2.2 • Instaliranje jezika Python 3 22
2.3 • Početak programiranja 22
2.4 • IDLE 24
2.5 • Interaktivna sesija 24
2.6 • Tipovi podataka 25
2.6.1 • Celi brojevi 25
2.6.2 • Brojevi sa pokretnim zarezom 25
2.6.3 • Stringovi i pisanje i čitanje datoteka 26
2.6.4 • Liste 36
2.6.4.1 • Razdvajanje stringova u listama stringova 39
2.6.5 • Rečnici 40
2.6.6 • N-torke 42
2.7 • Funkcije 42
2.8 • Uvozi 44
2.9 • Logički operatori 46
2.10 • Bit i bajtovi 46
2.12 • Heksadecimalna notacija 49
2.13 • Postojanost objekata 50

2.14 • Kompajlirane Python datoteke 51
2.15 • Unicode stringovi 51
2.16 • Bajtovi i nizovi bajta 54
2.17 • Kodiranje i dekodiranje bajtova i Unicode-a 56
2.18 • Nazivi promenljivih 57

POGLAVLJE 3 • GRAFIČKI KORISNIČKI INTERFEJSI 59

3.1 • Uvod 59
3.2 • Tkinter 59
3.2.1 • Kliknite za pokretanje programa 60
3.3 • Dodavanje vidžeta 61
3.4 • Povratni pozivi 61
3.5 • Opcije raspoređivača elemenata 62
3.5.1 Jednostavan raspored 62
3.5.2 Ugnežđeni raspored elemenata 64
3.6 • Unos i ispis podataka pomoću vidžeta 65
3.6.1 • Scale i Label vidžeti 66
3.6.2 • Unos 67
3.6.3 • Kontrolne promenljive 68
3.6.4 • Polja za potvrđivanje 68
3.6.5 • Dugmad za izbor 69
3.6.6 • Canvas vidžet 70

POGLAVLJE 4 • OBJEKTNO-ORIJENTISANO PROGRAMIRANJE 73

4.1 • Uvod 73
4.2 • Klasa 73
4.3 • Nasleđivanje 75
4.4 • Rezime objektne orijentacije (OO) 79

POGLAVLJE 5 • ARDUINO UNO KAO SLAVE UREĐAJ 81

5.1 • Uvod 81
5.2 • Master-slave dizajn 81
5.2.1 • USB serijski interfejs 81
5.2.2 • Otpremanje Arduino programa 81
5.2.3 • Konfigurisanje Arduina pomoću adrese i ID projekta 82
5.3 • Klasa Uno 84
5.3.1 • Kreiranje instanci slave programa 85

5.3.2 • Upisivanje u digitalnim izlaznim pinovima 85
5.3.4 • Upisivanje u analognim izlaznim pinovima.....87
5.3.5 • Čitanje iz analognih ulaznih pinova.....87
5.3.6 • Rezime IO funkcija.....87
5.4 • Praktični primeri u kojima se koriste hardveri 88
5.4.1 • Upisivanje u digitalnim izlazima 88
5.4.2 • Upisivanje u analognim izlazima 91
5.4.3 • Očitavanje digitalnih ulaza 92
5.4.4 • Očitavanje analognih ulaza 93
5.5 • Dijagram stanja slave uređaja i komunikacioni protokol 95
5.5.1 • Dijagram stanja slave uređaja 95
5.5.2 • Kontrola prenosa podataka 96
5.5.3 • Formati telegrama 97
5.6 • Python kraj komunikacionog protokola 99

POGLAVLJE 6 • DODATNI PRIMERI GUI-JA.....101

6.1 • Uvod.....101
6.2 • Klasa za crtanje Graph 101
6.2.1 • Prosleđivanje argumenata funkciji bez određivanja broja argumenata (*args) 102
6.2.2 • Prosleđivanje argumenata funkciji pomoću argumenata ključnih reči.102
6.2.3 • Povezivanje događaja 103
6.2.4 • Modul __name__ 103
6.2.5 • Sledeći kod 105
6.3 • Eksperimenti punjenja/pražnjenja kondenzatora 109
6.4 • Iscrtavanje karakteristika tranzistora 111
6.4.1 • Izvršavanje više niti 116

POGLAVLJE 7 • GRAFIKONI BITMAPE.....119

7.1 • Uvod.....119
7.2 • Datoteka bitmape 119
7.3 • Čitanje i pisanje binarnih datoteka 121
7.4 • Slika 122
7.5 • Modifikovana slika 124
7.6 • Zaglavlje 125
7.6.1 • Modul struct 126

7.7 • Neželjeni bajtovi....128
7.8 • Klasa BmpDraw....128
POGLAVLJE 8 • PROJEKAT ZA PRAĆENJE POTROŠNJE ELEKTRIČNE ENERGIJE....131
8.1 • Uvod....131
8.2 • Sonda i daljinski senzor....131
8.3 • PIC mikrokontroler....133
8.3.1 • Arduino Slave kao uređaj za programiranje PIC-a....134
8.4 • Radio veza i Arduino slave....135
8.5 • Pokretanje sistema i desktop programa....135
8.5.1 • Desktop program za kalibraciju....136
POGLAVLJE 9 • INTERFEJSI VEB PREGLEDAČA....145
9.1 • Uvod....145
9.2 • HTML 5....145
9.3 • CherryPy Python veb radni okvir....146
9.3.1 • Korisnički unos pomoću veb obrazaca....148
9.4 Ampermetar i veb aplikacija trenda potrošnje struje....150
9.4.1 • Uno slave program....150
9.4.2 CherryPy program....151
9.5 • Raspberry Pi kao veb server....157
9.5.1 • Konfigurisanje Wi-Fi-ja iz komandne linije na Pi-ju....158
9.5.2 • Pokretanje Python programa automatski nakon ponovnog podizanja sistema....159
9.6 • Pristup sa Interneta....161
9.7 • Prilagođavanje ovog projekta....162
POGLAVLJE 10 • CURSES INTERFEJSI....163
10.1 • Uvod....163
10.2 • Početak rada sa meračima....163
10.2.1 • Funkcija wrapper....164
10.2.2 • Windows na stdscr-u....165
10.2.3 • Boja....165
10.3 • Prilagođeni vidžeti....169
10.3.1 • Indikatorska lampica....169
10.3.2 • Analogni vidžet merača....172
10.3.3 • Kombinovanje atributa pomoću operatora nad bitovima OR....173
10.3.4 • Događaji miša za unos korisnika....173

10.3.5 • Vidžet točkića 174
10.3.6 • Vidžet dugmeta.....178
10.3.7 • Vidžet natpisa.....179
10.4 • Modul sa svim klasama.....180
DODATAK A • DODATAK A • IZVORNI KOD ZA ARDUINO UNO SLAVE 185
DODATAK B • PYTHON UNO SLAVE.....193
B.1 • Pydoc 199
DODATAK C • ARDUINO SLAVE PROGRAMSKA ALATKA.....201
DODATAK D • IZVORNI KOD ARDUINO SLAVE ALATKE ZA POSTAVLJANJE ID-IJA....203
DODATAK E • IZVORNI KOD ZA PIC.....207
DODATAK F • PIC PROGRAMSKA ALATKA.....209
F.1 • 12F1622_PROGRAMMER_V1.0.py.....209
DODATAK G • MODUL CURSES VIDŽETA 215
G.1 • Curses_widgets.py 215

Predgovor drugog izdanja

Ova knjiga je namenjena da bude praktičan vodič za povezivanje jednostavnih hardverskih projekata sa računarom. U knjizi su obuhvaćene mnoge teme. One nisu slučajan izbor, jer svaka od njih je potrebna da bi se postigla komunikacija sa određenim uređajem interfejsa, koji je izabran zbog njegove dostupnosti i niske cene.

Uređaj je već spremni Arduino Uno. Pošto se u ovoj knjizi razmatra Python, obezbeđen je Arduino Sketch koji omogućava čitaocu da koristi Uno-e kao slave uređaje. Python programi rade na Linux računarima da bi se kontrolisala spoljna elektronska kola kroz ulaze i izlaze Uno-a. U svaki projekat može da se uključi više ovih slave uređaja, jer svakom od njih se dodeljuje jedinstvena adresa.

Do kraja knjige bićete u stanju da kontrolišete i pratite hardver sa vašeg grafičkog korisničkog interfejsa (GUI) ili veb pregledača preko Interneta. Da biste to uradili, u knjizi će biti objašnjeno programiranje u jeziku Python, pisanje veb stranica i upotreba Python-a kao veb servera. Biće predstavljena neka elektronska kola koja ćete moći da integrišete sa softverskim projektima. Jedan od projekata će biti praćenje trenutne potrošnje energije u zgradi ili kući. U drugom projektu će biti prikazane karakteristike tranzistora.

Takođe je dat uvod u Linux. Izabrana je Debian distribucija, ali bilo koja od njenih izdanja, kao što je Ubuntu, Mint ili Raspbian, je sasvim u redu ako hoćete da koristite Raspberry Pi. U prvom izdanju knjige korišćen je prilagođeni live Linux CD koji je sadržao sve programe i potrebne softvere. Međutim kada je štampano prvo izdanje, ova distribucija je zastarela, pa se preporučuje da uvek koristite najnoviju distribuciju. Programe za vežbanje možete preuzeti na Elektorovom veb sajtu ili na veb sajtu infoelektronika.net. Debian je izabran zbog jednostavnog instaliranja aplikacija, stabilnosti i popularnosti.

Treba imati na umu da ni jedna tema nije detaljno obrađena. Međutim, obezbeđeno je dovoljno informacija da se postigne cilj i da se čitaocu omogući polazna osnova za svaku temu. Nove teme koje se razmatraju u svakom poglavlju neće biti pokrivene u detalje, ali u narednim poglavljima te teme će biti dopunjene po potrebi. Na Internetu je dostupna ogromna količina dokumentacije. Problem je u tome što treba znati odakle početi, jer za praktične projekte potrebno je naučiti veliki broj tema. Ova knjiga će vam obezbediti početni nivo koji možete da nadograđujete.

Pretpostavlja se da čitalac zna da koristi računar, ali da ne poznaje programiranje na očekivani način. Potrebno je osnovno znanje o električnoj energiji i zasebnim elektronskim komponentama, kao što su otpornici, kondenzatori, prekidači i LED-ovi. Biće korisno ako znate da lemite, ali projekti se mogu isprobati pomoću prototipskih ploča bez lemljenja. Prototipske ploče mogu da budu jeftinije, zato što se komponente i žice mogu ponovo koristiti. Takođe, konstruisanje je mnogo brže.

Ovo je drugo izdanje knjige u kome postoje četiri velike izmene: koristi se Python 3, umesto Python 2, ploča Arduino Uno je ciljni hardverski interfejs umesto razvojne ploče FTDI FT245, koristi se standardna Debian distribucija i umesto upotrebe Apache veb servera, koristi se veb okruženje Python za opsluživanje veb stranica.

Nadam se da će vam knjiga biti zanimljiva i zabavna.

Poglavlje 1 ● Linux

1.1 ● Uvod

Linux je izabran kao OS (operativni sistem), jer ćete videti dok budete čitali knjigu da ćete slobodno moći da menjate mnogo toga i da eksperimentišete. Ako ste već iskusni korisnik Linux-a i navikli ste da koristite komandnu liniju, onda će vam sadržaj ovog poglavlja već biti poznat. Možda ste koristili samo grafički korisnički interfejs koji Linux sadrži ili možda ste sasvim novi u Linux-u. Primeri u knjizi su razvijeni pomoću verzije Debian 8 Jessie, koja je najnovija stabilna verzija. Postoji nekoliko distribucija koje se zasnivaju na Debian-u. Dve popularne verzije su Mint i Ubuntu, a ako želite da koristite Raspberry Pi, tu je Raspbian.

1.2 ● Pribavljanje operativnog sistema

Pretpostavljam da imate OS zasnovan na Debian-u ili želite da ga instalirate. Ako to nije slučaj, možete da koristite virtuelnu mašinu, kao što je VirtualBox, koja će vam omogućiti da zadržite vaš postojeći OS, ali može da radi sporije nego što je to poželjno. U vreme pisanja ove knjige, Debian se mogao preuzeti na adresi *https://www.debian.org/distrib/netinst* i njegovo instaliranje je lako.

1.3 ● Instrukcije

Ako ste novi u Linux-u, izgledaće vam čudno na prvi pogled, posebno direktorijumi datoteka. U knjizi ćemo koristiti komandnu liniju da bismo pristupili datotekama i pokrenuli aplikacije i tako dalje. Ovo može da izgleda kao korak unazad, ali Linux komandna linija je puna brzih prečica i trikova koji će vam uštedeti vreme. Takođe, dok budete pisali programe, moraćete ponekad da iste naredbe komandne linije stavite u kod.

Na desktopu ili u meniju aplikacija u vašem OS-u pronađite ikonu za „Emulator terminala".



Slika 1-1 Emulator terminala

Na slici 1-1 prikazano je da se nalazimo u matičnom direktorijumu - znak ~ označava matični direktorijum. Ovo je matični direktorijum korisnika pod nazivom „user".

Kao uvod u komandnu liniju, ukucajte „cd /" bez navodnika. Pritisnite return, zatim ukucajte „ls" i ponovo pritisnite return. Rezultat je prikazan u nastavku. Direktorijumi se prikazuju plavom bojom. Postoji mnogo komandi za različite zadatke i brzo ćete naučiti one uobičajene. Komanda „cd" se koristi za promenu direktorijuma, a „ls" za prikaz sadržaja direktorijuma.

Poglavlje 2 • Početak rada u jeziku Python

2.1 • Uvod

Mikroprocesori koriste brojeve. Ovi brojevi se fizički prikazuju pomoću binarnog sistema, koji je implementiran kao naponi. Kada računar obrađuje tekst, to je i dalje obrada brojeva. Na primer, znak „A" je broj 65 u standardnom američkom kodu za razmenu informacija (ASCII). Jedna od razlika između jezika Python 2 i 3 je način na koji se tekst kodira. ASCII je predstavljao način na koji je tekst podrazumevano kodiran u jeziku Python 2. Python 3 koristi Unicode, a razlike ćemo razmatrati na kraju poglavlja. Unicode je veoma važna tema koju treba da razumete.

Uobičajeno je da kod mikroprocesora može da napiše čovek u mašinskom kodu koji su brojevi, a koji predstavljaju instrukcije i podatke. Poboljšanje je u tome što može da se koristi asembler – program koji prihvata skraćenice za mašinski kod koje je napisao čovek i da piše datoteku koja je mašinski kod. Program 2-1a na levoj strani je kratak program koji je napisan za asembler FASM.

Program 2-1 Primer asemblera i „C" kodiranja

```
;prog_02_01.asm

format ELF executable
jmp main
message  db   "It's a lot easier in Python!",10
len equ $-message

main:
mov edx, len
mov ecx, message
mov eax, 4
int 0x80

mov eax, 1
int 0x80;
```

Prog 2-1a (prog_0201.asm)

```
//prog_02_01.c

#include<stdio.h>

int main(){

printf("It's a lot easier in python\n");

return 0;

}
```

Prog 2-1b (prog_02_01.c)

Izvorni kod prog_02_01.asm je asembliran u izvršivu datoteku assembler_demo i nalazi se u zip datoteci. Ako želite, možete da izvršite datoteku u komandnoj liniji na sledeći način (pobrinite se da ste u istom direktorijumu u kojem je datoteka, a zatim ukucajte ./

assembler_demo, u stvari, tabulator ./a će automatski dovršiti komandu):

```
user@debian:~/programs/chap02_progs$ ./assembler_demo
It's a lot easier in python!
user@debian:~/programs/chap02_progs$
```

Asembler se još uvek koristi kada je važna maksimalna brzina ili u aplikacijama, kao što su mali mikrokontroleri sa ograničenom memorijom. Kod nije prenosiv na druge platforme i programer treba da poznaje arhitekturu mikroprocesora i druge detalje o hardveru.

Program 2-1b radi isto ono što i asemblerski program tako što koristi jezik C. C koristi program pod nazivom kompajler koji vam omogućava da pišete u formi koja je čitljivija i bavi se detaljima hardvera. Kompajler preuzima C izvorni kod koji je na-

pisao čovek i kompajlira ga u datoteke koje može da izvrši procesor pomoću vašeg operativnog sistema.

Ako je to ono što radi C kompajler, zašto nam je onda potreban Python? Uglavnom, Python se može mnogo lakše koristiti od jezika C. Python je interpretiran jezik. Interpretator Python je višeplatformski jezik. Python preuzima vaš skript i kompajlira ga u nešto što se zove kod bajta, koji radi na virtuelnoj mašini na vašem operativnom sistemu – ovo kompajliranje se izvršava kada pokrenete program. Ovo je sporije od kompajliranog C programa, ali možda nije sporo za ono što radite. Python je dovoljno brz za aplikacije iz ove knjige.

U Pythonu-u gorenavedeni programi izgledaju ovako:

```
print (“It’s a lot easier in Python!”)
```

Ako ste novi u Python-u 3, ali imate neko iskustvo u Python-u 2 primetićete da su se neke stvari promenile – na primer, iskaz "print 'Hello World'" je sada funkcija i napisana je kao "print('Hello World')", ali postoje i druge razlike.

2.2 ● Instaliranje jezika Python 3

U ovoj knjizi se koristi Python 3. Iako je Python 2 još uvek zastupljen, ako ste novi, treba da koristite Python 3. Prvo proverite da li je Python 3 već instaliran tako što ćete ukucati „python3" u komandnu liniju.

```
user@debian:~$ python3
Python 3.4.2 (default, Oct  8 2014, 13:14:40)
[GCC 4.9.1] on linux
Type “help”, “copyright”, “credits” or “license” for more informati
>>>
```

Time ćete ući u interaktivnu sesiju sa Python-om, čime ćemo se baviti kasnije. Da biste izbegli upotrebu komande „Ctrl d", tj. da biste izbegli da istovremeno pritisnete tastere Control i d, pobrinite se da imate Python 3. Ako nemate, možete ga lako instalirati. Dok ste povezani sa Internetom, ukucajte ovo u komandnu liniju:

```
user@debian:~$ sudo apt-get install python3
```

Jedna od zaista dobrih funkcija u Linux OS-u zasnovanom na Debian-u je „menadžment paketa" koji veoma olakšava instalaciju i uklanjanje softvera. Nisu sve Linux distribucije ovoliko jednostavne, posebno Slackware. Iako je Slackware moja omiljena distribucija, može da zahteva mnogo više posla, jer morate ručno da pronađete i instalirate sve druge softvere od kojih zavisi softver koji instalirate. Ove „zavisnosti" mogu da se pojave na nekoliko nivoa.

2.3 ● Početak programiranja

Otvorite Mousepad tako što ćete istovremeno pritisnuti tastere Alt i F2, a zatim ukucajte „mousepad". Međutim, možete da koristite uređivač teksta koji vi želite. Unesite sledeći program (Primetite da su druga i treća linija identične – ovo je deo logike koda. Iako je kod ponuđen, kucanje nekih programa će vam pomoći da se upoznate sa jezikom):

Poglavlje 3 • Grafički korisnički interfejsi

3.1 • Uvod

Što se tiče funkcionalnosti, odgovarajući programi mogu da se napišu samo pomoću komandne linije kao interfejs. Međutim, grafički interfejs je mnogo privlačniji, jer nudi uobičajene vidžete koje većina korisnika računara koristi – dugmad, polja za potvrđivanje, polja za unos teksta itd. Python omogućava jednostavan način za kreiranje ovih interfejsa za vaše programe. U dva projekta koja će biti prikazana kasnije u knjizi iscrtaćemo grafikon vrednosti koje su izvedene iz analognih veličina koje se mere. U poslednjem poglavlju prikazaćemo vam još jedan način za kreiranje GUI-ja za platforme koji imaju manje resurse i kojima ne treba X server.

Kao što smo napomenuli ranije, nećemo razmatrani ni jednu temu detaljno, ali umesto toga, čitaocima ćemo ponuditi skup uvodnih alatki za izradu projekata koji funkcionišu i dati uputstva za dalje učenje. Ovde su predstavljeni neki jednostavni primeri, koje ćete najbolje razumeti ako ih pokrenete i eksperimentišete sa njima.

3.2 • Tkinter

Tkinter je standardna biblioteka koja omogućava da se Python poveže sa grafičkom programskom bibliotekom Tk. Tkinter pruža veliku raznovrsnost. Predstavio sam nekoliko primera programa koji svakako ne obuhvataju sve ono što Tkinter nudi. Preporučio bih da pokrenete primere programa i prilagodite ih onako kako vama odgovora. Vidžeti koji čine interfejs imaju atribute sa određenim nazivima, a samo neki su prikazani u primerima programa.

Prvo treba da proverite da li već imate instaliran tkinter. Pokrenite Python 3 interaktivnu sesiju i ukucajte „import tkinter". Ako nije instaliran, dobićete sledeći rezultat:

```
user@debian:~$ python3
Python 3.4.2 (default, Oct  8 2014, 13:14:40)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tkinter
Traceback (most recent call last):
 File "/usr/lib/python3.4/tkinter/__init__.py", line 39, in <module>
    import _tkinter
ImportError: No module named '_tkinter'
```

Prilikom obrade prethodnog izuzetka, pojavio se još jedan izuzetak:

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/usr/lib/python3.4/tkinter/__init__.py", line 41, in <module>
raise ImportError(str(msg) + ', please install the python3-tk package')
ImportError: No module named '_tkinter', please install the python3-tk package
>>>
```

Da biste instalirali tkinter, ukucajte sledeće u komandnu liniju dok ste povezani sa Internetom:

```
user@debian:~$ sudo apt-get install python3-tk
```

Počinjemo tako što ćemo kreirati najjednostavniji grafički program koji kreira prazan prozor, a zatim ne radi ništa drugo, osim što menja veličinu, smanjuje, povećava i zatvara prozor. Ove tri linije koda, u stvari, obavljaju mnogo posla.

Program 3-1 Prazan prozor

```
#prog_03_01

import tkinter

root = tkinter.Tk()      # Tk() is the top level class.
root.mainloop()         # Mainloop is the  loop that responds to events like button clicks.
```

U ovom programu postoje samo tri linije koda, ali postoje i tri nove stvari na koje treba da ukažemo. Prvo, uvozimo „tkinter" – paket koji nam treba da bi kreirali GUI-ije.

Drugo, linija „root = tkinter.Tk()" je primer kreiranja instance klase, koje ćemo detaljnije razmatrati u poglavlju o objektno-orijentisanom programiranju (OOP). Ovde je „root" naziv koji je, u ovom slučaju, dodeljen instanci klase Tk, ali vi možete da izaberete naziv po vašoj želji. Ako ovo uporedimo sa stvarnim svetom, vi ste instanca klase ljudi i imate ime. Treća linija „root.mainloop()" znači da nakon što je kreirana instanca „root", pozivamo funkciju „mainloop() ili metod koji pokreće petlju vođenu događajem koja kontroliše interfejs. Pritisak na dugme je događaj i treba reagovati na događaj tako što se poziva neka funkcija u programu.

Sada ćemo napisati jedan veoma kratak primer GUI-ja da biste videli kako se dodaju vidžeti, pozivaju funkcije kao reakcija na događaj i kako se kontroliše raspored elemenata i izgled. Različiti vidžeti će biti prikazani kako budemo napredovali.

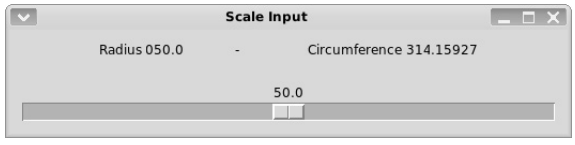
3.2.1 ● Kliknite za pokretanje programa

Do sada su naši programi pokrenuti pomoću instrukcije komandne linije „python3 prog_xxx.py". Možemo da izvršimo skript tako što ćemo upotrebiti instrukciju komandne linije „chmod 755 prog_xxx.py". Ovo će promeniti dozvole za datoteku (pogledajte Poglavlje 1.8). Da bi se skript pokrenuo, dodata je prva linija „#!/usr/bin/env python3". Iako ovo izgleda kao komentar, počinje znakom „!" iza kojeg sledi putanja Python interpretatora. Kada je postavljena u prvoj liniji, Linux zna da treba da pozove „python3" da bi se skript izvršio. Možete da pokrenete program iz komandne linije pomoću „./prog_03_02.py" the "./" (što znači da se datoteka traži u trenutnom direktorijumu) ili možete dva puta da kliknete datoteku u File Manager-u. Drugi način je da kopirate izvršivši skript na desktop sa kojeg možete da ga pokrenete tako što ćete dva puta kliknuti na taj skript. Preporučujem da koristite komandnu liniju kada otklanjate greške u programu.

3.6.1 • Scale i Label vidžeti

Scale vidžet se može koristiti kao interfejs za unošenje i ispisivanje podataka. Posebno je koristan za kontrolne aplikacije, jer „imitira" tradicionalne analogne kontrolne uređaje. Sledeći program, Program 3-7 predstavlja Label vidžet, koji korisniku prikazuje tekst koji se može samo čitati. U programu je kreirana skala koja se naziva 'scale_in'sa opcijama i postavljena je horizontalno, dugačka je 500 piksela, ima rezoluciju 0.1 i poziva funkciju 'drive_output()' kada se pomera. Ono što se ne može uočiti kada se pregleda kod je da skala prosleđuje vrednost svoje pozicije funkciji koja se poziva, pa tako u funkciji imamo argument pod nazivom 'scale_value', ali može da ima bilo koji važeći naziv. Ova promenljiva je string koji sadrži vrednost pozicije u obliku broja sa pokretnom tačkom. Vrednost pozicije skale se pretvara u broj sa pokretnom tačkom koja je dodeljena lokalnoj promenljivoj r. Tekst za natpis je formatiran da bi imao pet decimalnih mesta u opsegu izlaza i minimalnu širinu uključujući i decimalnu tačku od devet znakova. Poluprečnik izlaza ima jedno decimalno mesto i širinu od pet znakova. Poluprečnik i širina su popunjeni vodećim nulama. Ovo omogućava bolji digitalni prikaz bez pomeranja pozicije. Znak „\t" umeće razmak između redova u string. Za izračunavanje vrednosti pi, uvezena je matematička biblioteka.

Program 3-7 Label vidžet




```
#!/usr/bin/env python3
#prog_03_07.py

import tkinter
import math

def drive_output(scale_value):
    r = float(scale_value)
    c = 2 * math.pi * r
    lbl_out.configure(text = 'Radius {0:05.1f} \t-\t Circumference {1:09.5f}'.format(r, c))

root = tkinter.Tk()

root.title('Scale Input')
scale_in = tkinter.Scale(orient = 'horizontal', length = 500, resolution = 0.1, command = drive_output)
lbl_out = tkinter.Label()
lbl_out.grid(row = 0, column = 0, pady = 10)
scale_in.grid(row = 1,column = 0, pady = 10, padx = 10)


root.mainloop()
```

Program 3-8 koristi još jednu skalu za prikaz izlazne vrednosti. Ovde se metod skale set() koristi za utvrđivanje pozicije klizača. Takođe, dodati su markeri „tickinterval" i opcija pod nazivom label, koja je deo objekta Scale.

Linija „commented out" je alternativni metod za direktno dobijanje pozicije skale kao broja sa pokretnom tačkom. Da biste to isprobali, premestite prefiks komentara # na drugu liniju. Definicija za 'scale_out' ne sadrži povratni poziv, jer se koristi kao vidžet za unos, a ne za ispis.

Linije koje kreiraju instance skale, scale_in i scale_out predstavljaju dugačke linije koda i preporučuje se da linije koje su ispisane u uređivaču ne sadrže više od 80 znakova. Možete da prelomite jednu liniju na više linija ako je deo koji se prelama zatvoren u zagradama.

Program 3-8 Skala za prikazivanje

```
##!/usr/bin/env python3
#prog_03_08.py

import tkinter
import math

def drive_output(scale_value):
    r = float(scale_value)
    #r = scale_in.get() # Can also be used
    c = 2 * math.pi * r
    scale_out.set(c)

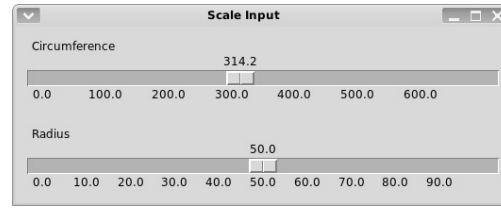
root = tkinter.Tk()
root.title('Scale Input')

scale_in = tkinter.Scale(orient = 'horizontal', length = 500, resolution = 0.1,
                         command = drive_output, label = 'Radius', tickinterval = 10)

scale_out = tkinter.Scale(orient = 'horizontal', length = 500, resolution = 0.1,
                          label = 'Circumference', tickinterval = 100, to = 700)

scale_out.grid(row = 0,column = 0, pady = 10, padx = 10)
scale_in.grid(row = 1,column = 0, pady = 10, padx = 10)

root.mainloop()
```



3.6.2 • Unos

Ako želite da direktno unesete podatke, Entry vidžet može da se koristi za unos jedne linije teksta.

Program 3-9 koji je dat u nastavku koristi tekstualni unos kao numeričku vrednost. Kada se klikne na dugme, funkcija Calculate() pretvara string u Entry vidžetu u broj sa pokretnom tačkom i stavlja odgovor na natpis. Ovde ne postoji zaštita za nenumeričke podatke koji se unose. Možete da napišete kod koji će postaviti string u polju za unos pomoću metoda insert(index, s), u kome je s string, koji se umeće u polje za unos pre indeksa pozicije.

Program 3-9 Unos teksta

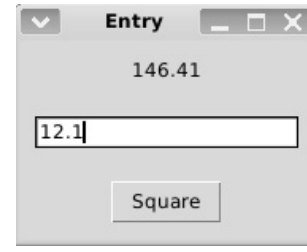
```
#!/usr/bin/env python3
#prog_03_09

import tkinter

def calculate():
    x_string = ent_input.get()
    x = float(x_string)
    y = x*x
    y_string = str(y)
    lab_output.configure(text = y_string)

root = tkinter.Tk()
root.title('Entry')
ent_input = tkinter.Entry(root)
ent_input.insert(0,'0.0')

lab_output = tkinter.Label(root,text = '0.0')
```



Poglavlje 4 • Objektno-orijentisano programiranje

4.1 • Uvod

Do sada su naši programi bili ono što se zove proceduralni programi. Objektna orijentacija je drugačiji način pisanja programa. Reč je o kreiranju softverskih modela stvarnih stavki kao odvojenih objekata. Ovi objekti sadrže podatke i funkcije koji su povezane sa tim objektima. Imajte na umu da je obrada ove teme sasvim dovoljna da biste napravili radne interfejse. Predstavićemo nove koncepte, klase i nasleđivanje. Objekti o kojima je ovde reč su pojedinačne instance klase. Vi ste instanca ljudske klase. Imate ime po kome se razlikujete od drugih instanci ljudske klase. Ljudska klasa ima atribut height (visina) ako je vaše ime jsmith. Vaša visina može da bude „jsmith.height = 1.6".

4.2 • Klasa

Blok koda koji je klasa je kao fabrika koja proizvodi instance te klase. Ove instance enkapsuliraju podatke i metode. Metod je termin koji označava funkciju u klasama. Ovo je pogodan način programiranja, jer programi postaju sve komplikovaniji.

Sada ćemo razmotriti početni primer elektronskog kola naponskog komparatora.

Ako je ulazni napon iznad određenog nivoa, pozvaćemo zadatu vrednost, a zatim se izlaz postavlja na on, u suprotnom je postavljen na off.

U Pythonu možemo da kreiramo klasu da bismo predstavili naponske komparatore kao u Programu 4-1. Ovaj program ima klasu pod nazivom „Comparator". Klasa se koristi za kreiranje dve instance naponskog komparatora, koji ima različite zadate vrednosti.

Kreiranje klase počinjemo pomoću reči „class", a zatim ćemo joj dodeliti naziv, u ovom slučaju, Comparator. Pravilo je da nazivi klase počinju velikim slovom.

Uvlačenje ispod prve linije je ostatak klase. Naša klasa ima dva metoda.

Linija „comp1=...." posle kraja koda klase kreira instancu naponskog komparatora I koji sam nazvao comp1.

Prilikom kreiranja instance klase, metod __init__ () te klase se poziva automatski ako postoji. Reč init ima dve donje crte sa obe strane. Sada pogledajte ovaj metod: postoje dva argumenta – prvi argument je „self" koji označava da funkcija pripada ovoj instanci.

Program 4-1 Naponski komparatori

```
#!/usr/bin/env python3
# prog_04_01a.py

class Comparator:
    def __init__(self, setpoint):        # Method runs whenever a new instance is created
        self.switch_level = setpoint
        self.output = False
    def switching(self, voltage):
    if voltage > self.switch_level:
            self.output = True
    else:
            self.output = False          # End of class code
```

```
comp1 = Comparator(5)              # Creates instance comp1, calls method __init__
comp2 = Comparator(6)              # Creates instance comp2, calls method __init__

print('comp1 output is {}'.format(comp1.output))

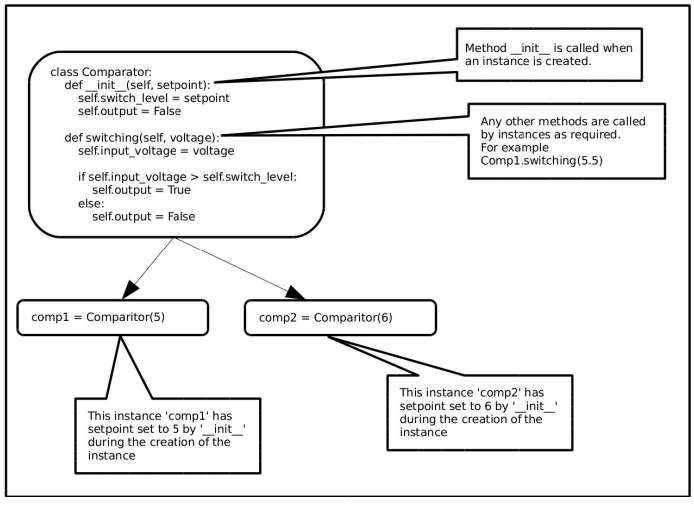
print('comp2 output is {}'.format(comp2.output))
```

Drugi parametar koji sam nazvao „setpoint" je argument koji se prosleđuje metodu __init__() kad god se kreira nova instanca. Ovde je vrednost 5. Sledeća linija koda postavlja promenljivu switch_level na vrednost 5, koja će biti zadata vrednost za komparator comp1. Ponavljam, self znači da ova promenljiva pripada instanci koja se kreira. Poslednja linija metoda __init__() postavlja izlaz promenljive na off ili false.

Linija „comp2=....." kreira drugu instancu klase Comparator, ali ovoga puta kada se pokrene metod __init__() setpoint se postavlja na vrednost 6.

Slika 4-1predstavlja slikovni prikaz kreiranja ove instance.

Ako pogledamo poslednje dve linije programa, videćemo da ove linije ispisuju stanje izlaza. Funkcija switching(self, voltage) još uvek nije pozvana.



Slika 4-1 Slikovni prikaz kreiranja instance

Pokretanjem Programa 4-1 u komandnoj liniji dobijamo sledeće:

```
user@debian:~$ ./prog_04_01a.py
comp1 output is False
comp2 output is False
user@debian:~$
```

Sada ćemo primeniti ulazni napon od 5.5 na naše dve instance klase Comparator.

Ovo je svrha metoda switching(self, voltage). Da biste pozvali metod switching() za komparator comp1 sa ulaznom vrednošću napona od 5.5 koristite „comp1.switching(5.5)" i za comp2 koristite „comp2.switching(5.5)". Program 4-2 prikazuje ove linije programskog koda i dve linije za ispisivanje novih izlaznih stanja.

Poglavlje 5 • Arduino Uno kao slave uređaj

5.1 • Uvod

U prvom izdanju ove knjige koristili smo UM245 USB interfejs proizvođača Future Technologies Development International (FTDI) za čitanje i pisanje jednobajtnih znakova u hardverski projekat. U ovoj knjizi ćemo koristiti Arduino Uno kao slave uređaj. Nije potrebno pisati kod za Uno, jer je obezbeđena Arduino Uno univerzalna skica. Ovo omogućava Arduinu da reaguje na programe koji su napisani u Python-u na računaru koji je master uređaj. Skica je dostupna kao datoteka hex koju uslužni program AVRDUDE može da učita u vaš Arduino. Python alatka je dostupna u programima.

U ovom poglavlju ćemo prikazati kako da programirate i konfigurišete Arduino i videćete kako da koristite komunikacione funkcije u Pythonu da biste upravljali slave uređajima. Na kraju ovog poglavlja videćete više detalja o protokolu, premda kasnije možete pročitati korisne informacije o otkrivanju grešaka.

5.2 • Master-slave dizajn

Programi koji su napisani u Python-u imaju GUI-je koje korisnik može da koristi da bi pratio i kontrolisao pravi elektronski hardver. Arduino Uno sadrži hardverski interfejs digitalnih i analognih ulaza i izlaza (IO), koji se mogu povezati sa elektronskim kolima. Arduino je slave uređaj, a računar na kome radi Python program je master. Za pouzdanu komunikaciju između Python programa i IO-a, dostupan je serijski komunikacioni protokol koji ima veliki broj funkcija koje prenose podatke u okvirima. Svaki okvir ima kontrolni zbir koji otkriva greške.

5.2.1 • USB serijski interfejs

Da bi Python programi mogli da komuniciraju sa slave uređajem, treba da instalirate Python biblioteku za serijsku komunikaciju. Ovo možete veoma jednostavno uraditi iz komandne linije, samo ukucajte sledeće:

```
user@debian:~$sudo apt-get install python3-serial
```

Za ovo je potrebno da koristite USB serijski interfejs. Međutim, treba da se pridružite grupi „dialout", jer u suprotnom nećete imati dozvolu da otvorite portove "/dev/ttyACM*".

Da biste se pridružili grupi „dialout", ukucajte sledeće u komandnu liniju:

```
user@debian:~$sudo usermod -a -G dialout user
```

Napominjemo da treba da koristite svoje korisničke ime umesto „user".

5.2.2 • Otpremanje Arduino programa

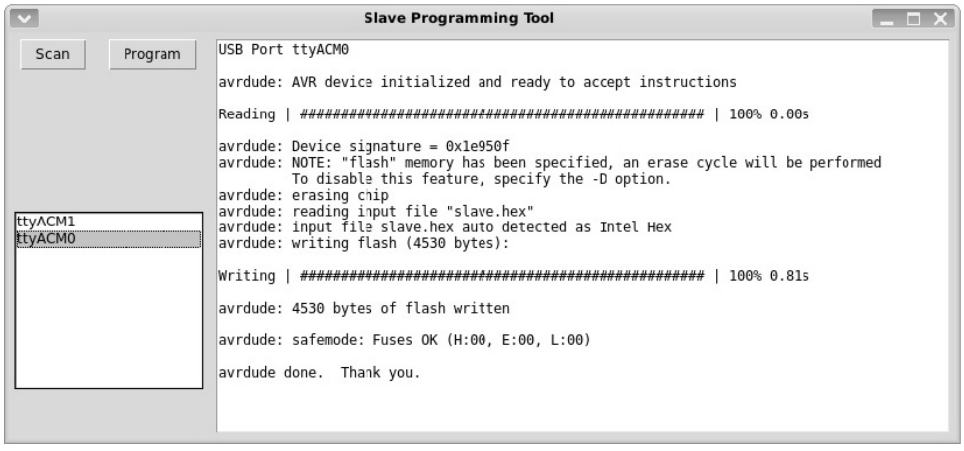
Arduino program (skica) je već upisana u Arduino C/C++ funkcije i kompajliran je u „.hex" datoteci koju možete da otpremite na vaš Arduino Uno.

Ovo možete uraditi pomoću AVRDUDE-a – projekta otvorenog koda, koji je napisao Brian S Dean. AVRDUDE je uslužni program komandne linije za koji je potrebno nekoliko argumenata komandne linije. Ovde ćemo predstaviti Python GUI alatku da bismo vam to olakšali. Na slici 5-1 prikazan je korisnički interfejs.

Instaliranje AVRDUDE-a možete lako obaviti tako što ćete u komandnu liniju ukucati sledeće:

```
user@debian:~$ sudo apt-get install avrdude
```

Kada to završite, onda možete koristiti Python GUI alatku.



Slika 5-1 Alatka za slave model programiranja

Da biste koristili alatku za programiranje, postupite na sledeći način:

Pokrenite program „arduino_slave_programming_tool_V1.0.py" u programima iz ovog poglavlja u zip datoteci. Pokrenite program bez Arduino USB kabla. Pobrinite se da pokrenete program iz ovog direktorijuma – pogledajte objašnjenje u nastavku.

Kliknite na dugme „Scan" i obratite pažnju na portove koji su prikazani u prozoru.

Priključite Arduino koji želite da programirate i ponovo kliknite dugme „Scan". Trebalo bi da je prikazan port koji koristi Arduino.

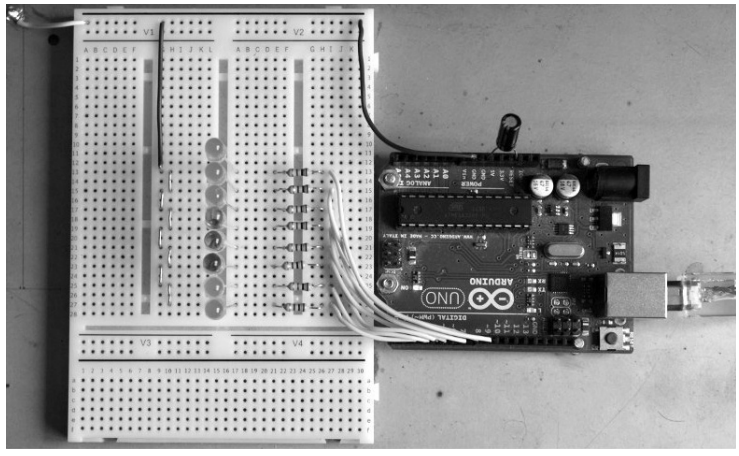
Izaberite ovaj port i zatim kliknite na dugme „Program". Trebalo bi da se pojavi izlazni prozor, kao što je prikazano na gornjoj slici.

Alatka za programiranje treba da pristupi hex datoteci, koja je poznata pod nazivom slave.hex. Ovo je kopija hex datoteke za aktivnu skicu, Arduino_Slave_17_03_26a.ino.standard.hex. Ako premestite alatku za programiranje u drugi direktorijum, morate da premestite i datoteku slave.hex.

5.2.3 ● Konfigurisanje Arduina pomoću adrese i ID projekta

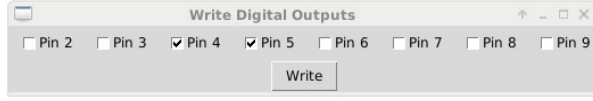
Možda imate više projekata, pa da biste sprečili slučajno pokretanje pogrešnog softvera sa hardverskim projektom, Arduinu se dodeljuje identifikacioni broj projekta. Master program na računaru će upravljati samo Arduinom koji ima ID projekta. Možda imate i više Arduina u projektu, pa je slave uređajima dodeljena adresa u okviru ovog projekta.

Ova dva broja se upisuju u električno izbrisivu programabilnu memoriju samo za čitanje (EEPROM) u Arduinu pomoću druge alatke koja se nalazi u zip datoteci. Adresa i brojevi projekta se zadržavaju u EEPROM-u kada se Arduino isključi, pa se unose samo jednom za svaki projekat.



Slika 5-9 Raspored za upisivanje u digitalnim izlazima

Program 5-1 Upotreba polja za potvrđivanje za selektovanje stanja izlaza




```
#!/usr/bin/env python3
#prog_05_01.py

import tkinter
import uno_slave

class GuiDigOuts():
    def __init__(self, master):
        self.uno1 = uno_slave.Uno(address = 1,project = 1, config_digital = [2,3,4,5,6,7,8,9])
        master.title('Write Digital Outputs')
        self.chkvar_2= tkinter.IntVar()        # Control variables
        self.chkvar_3= tkinter.IntVar()
        self.chkvar_4= tkinter.IntVar()
        self.chkvar_5= tkinter.IntVar()
        self.chkvar_6= tkinter.IntVar()
        self.chkvar_7= tkinter.IntVar()
        self.chkvar_8= tkinter.IntVar()
        self.chkvar_9= tkinter.IntVar()
        self.chkbtn_2 = tkinter.Checkbutton(master,text = 'Pin 2', variable = self.chkvar_2)
        self.chkbtn_3 = tkinter.Checkbutton(master,text = 'Pin 3', variable = self.chkvar_3)
        self.chkbtn_4 = tkinter.Checkbutton(master,text = 'Pin 4', variable = self.chkvar_4)
        self.chkbtn_5 = tkinter.Checkbutton(master,text = 'Pin 5', variable = self.chkvar_5)
        self.chkbtn_6 = tkinter.Checkbutton(master,text = 'Pin 6', variable = self.chkvar_6)
        self.chkbtn_7 = tkinter.Checkbutton(master,text = 'Pin 7', variable = self.chkvar_7)
        self.chkbtn_8 = tkinter.Checkbutton(master,text = 'Pin 8', variable = self.chkvar_8)
        self.chkbtn_9 = tkinter.Checkbutton(master,text = 'Pin 9', variable = self.chkvar_9)
        self.btn_write = tkinter.Button(master,text = 'Write', command = self.dig_write)

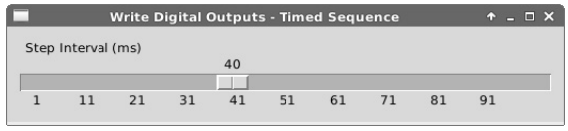
        self.chkbtn_2.grid(row = 0, column = 0, pady = 5, padx = 5)
        self.chkbtn_3.grid(row = 0, column = 1, pady = 5, padx = 5)
        self.chkbtn_4.grid(row = 0, column = 2, pady = 5, padx = 5)
        self.chkbtn_5.grid(row = 0, column = 3, pady = 5, padx = 5)
        self.chkbtn_6.grid(row = 0, column = 4, pady = 5, padx = 5)
        self.chkbtn_7.grid(row = 0, column = 5, pady = 5, padx = 5)
        self.chkbtn_8.grid(row = 0, column = 6, pady = 5, padx = 5)
        self.chkbtn_9.grid(row = 0, column = 7, pady = 5, padx = 5)
        self.btn_write.grid(row = 1, column = 0, columnspan = 8)
```

```
    def dig_write(self):
        self.uno1.write_digital_dict[2] = self.chkvar_2.get()
        self.uno1.write_digital_dict[3] = self.chkvar_3.get()
        self.uno1.write_digital_dict[4] = self.chkvar_4.get()
        self.uno1.write_digital_dict[5] = self.chkvar_5.get()
        self.uno1.write_digital_dict[6] = self.chkvar_6.get()
        self.uno1.write_digital_dict[7] = self.chkvar_7.get()
        self.uno1.write_digital_dict[8] = self.chkvar_8.get()
        self.uno1.write_digital_dict[9] = self.chkvar_9.get()
        self.uno1.write_digital()

tk = tkinter.Tk()
front_end = GuiDigOuts(tk)
tk.mainloop()
```

Program 5-2 je za isto kolo. Pokreće jedan LED tako da se pozicija svetlećeg LED-a pomera nazad/napred. Brzina promene položaja se kontroliše pomoću skale. Ovaj program predstavlja korisniku tehniku za pozivanje metoda u intervalima. Gornji prozori koji su, u ovom slučaju, naš master, imaju funkciju pod nazivom self.master.after(), koja poziva funkciju nakon zadatog vremena. Ovde funkciju koristimo na kraju metoda drive() da bismo ponovo pozvali ovaj metod. Vreme se zadaje pomoću skale. Za razliku od metoda time.sleep(), metod after() ne blokira ili drugim rečima, ne prekida tok programa. Važno je zapamtiti da metod sleep() ima zadato vreme izraženo u sekundama tako što koristi broj sa pokretnom tačkom, a metod after() ima zadato vreme izraženo u milisekundama tako što koristi ceo broj.

Program 5-2 Pozivanje metoda u intervalima




```
#!/usr/bin/env python3
#prog_05_02.py

import tkinter
import uno_slave
import time

class GuiDigOuts():
    def __init__(self, master):
        self.master = master
        self.uno1 = uno_slave.Uno(address = 1, project = 1, config_digital = [2,3,4,5,6,7,8,9])
        self.master.title('Write Digital Outputs - Timed Sequence')
        self.rate_scale = tkinter.Scale(master, orient = 'horizontal', length = 500, from_ =  1,
                            to = 100, tickinterval = 10,label = 'Step Interval (ms)')

        self.uno1.write_digital_dict = {2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0,}
        self.step = 2
        self.direction = 1
        self.rate_scale.grid(row = 0, column = 0, padx = 10,pady = 10)
        self.drive()

    def drive(self):
        self.uno1.write_digital_dict[self.step] = 0
        if self.step == 9:
            self.direction = -1
        if self.step == 2:
            self.direction = 1

        self.step = self.step + self.direction
        self.uno1.write_digital_dict[self.step] = 1
        self.uno1.write_digital()
        delay = self.rate_scale.get()
        delay = int(delay)
```

Poglavlje 6 • Dodatni primeri GUI-ja

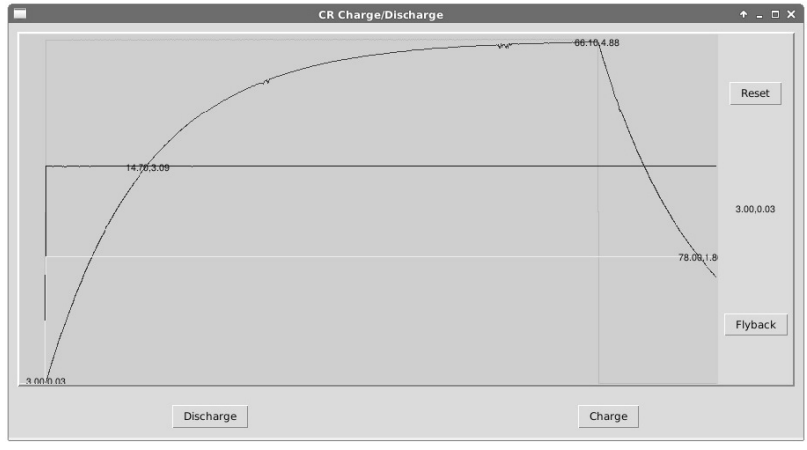
6.1 • Uvod

U ovom poglavlju prelazimo na naprednije GUI-je i predstavljamo dodatne aspekte Python-a i Tkinter-a. Reč je o GUI-jima za povezivanje sa Arduno slave. Morate da pročitate Poglavlje 5 pre nego što isprobate projekte u ovom poglavlju da biste razumeli kako se Arduino koristi ovde. Programi koji su predstavljeni će biti objektno-orijentisani. Programi će postajati duži i složeniji, ali objektno-orijentisano programiranje će olakšati pisanje koda. Ako ne možete da razumete delove koda, probajte da eksperimentišete tako što ćete menjati deo po deo koda, što će vam pomoći da ga razumete. Takođe, predstavićemo vam još Python funkcija.

6.2 • Klasa za crtanje Graph

Kao što je prikazano u Poglavlju 3, postoji ogroman broj standardnih vidžeta, ali treba da napišete kod da bi ti vidžeti radili ono što vi želite. Za nešto slično je često potrebno više od jednog programa i kucanje istog koda više puta, što je gubljenje vremena. Jedan takav primer je crtanje grafikona vrednosti analognih ulaza u odnosu na zadato vreme. Sledeći program je klasa koja može da se koristi za kreiranje instanci objekta koji će crtati grafikone i zahteva veoma malo koda koji ćete uvrstiti u druge projekte. Klasa se zove Graph i nalaziće se u modulu ili u zasebnoj datoteci.

Na slici 6-1 prikazan je GUI koji će biti Program 6-2, a koji je instanca klase Graph, i možete da vidite prilagođeni vidžet sa podlogom za crtanje i dva dugmeta Reset i Flyback sa zadebljanom ivicom. Ovaj vidžet možete jednostavno da postavite u svaki GUI koji napišete, tako što ćete da uvezete modul plotter, kreirate instancu Graph i pozovete njegove metode. Program 6-1 koji je dat u nastavku je plotter.py. Klasa ispisuje u obliku grafikona istovremeno do šest vrednosti analognih ulaza u odnosu na zadato vreme. Merenja na grafikonu mogu da se obave tako što ćete pritisnuti levi taster miša i ukazati na poziciju na grafikonu. Dugme Reset, takođe, ponovo počinje od levog kraja, ali i briše stavke sa podloge za crtanje. Osim što će vam uštedeti vreme, ovaj pristup omogućava da se pouzdano oslonite na kod, koji treba samo da uvezete i upotrebite.



Slika 6-1 GUI koji je kreirao Program 6-2

6.2.1 ● Prosleđivanje argumenata funkciji bez određivanja broja argumenata (*args)

Do sada smo određivali broj argumenata koji se prosleđuju funkciji – pogledajte pasus 2.7. Pretpostavimo da želimo da prosledimo različiti broj argumenata u zavisnosti od aplikacije. U Programu 6-1 obratite pažnju na definiciju funkcije „def plot (self, *args)". Ovde možemo da prosledimo do 6 vrednosti analognih ulaza koje će biti ispisane u obliku grafikona. U funkciji args je n-torka koja sadrži sve argumente koji su prosleđeni iz pozvane funkcije. Reč args nije obavezna, ali je pravilo. Ono što je važno je da se ispred reči args nalazi znak *.

Ovo je primer koji možete da isprobate u interaktivnoj sesiji.

```
>>> def fun(*args):
... print(args)
...
>>> fun('first_argument', 100.05, 99, 'last_argument')
('first_argument', 100.05, 99, 'last_argument')
>>>
```

Funkcija fun() se poziva sa četiri argumenata: prvi je string, drugi je broj sa pokretnom tačkom, treći je ceo poziv i četvrti je još jedan string. U definiciji funkcije koristi se *args. Funkcija ispisuje args i možete videti da je to n-torka. Ako upotrebimo ugrađeni metod enumerate() možemo da dobijemo indeksne brojeve i vrednosti argumenata koji su prosleđeni bez obzira na njihov broj. Funkcija enumerate() sadrži dve vrednosti: jedna je vrednost nabrajanja pozicije argumenta i druga je vrednost argumenta.

```
>>> def fun(*args):
... print(args)
... for i, v in enumerate(args):
...     print('argument {0:d} = {1:s}'.format(i,str(v)))
...
>>> fun('first_argument', 100.05, 99, 'last_argument')
('first_argument', 100.05, 99, 'last_argument')
argument 0 = first_argument
argument 1 = 100.05
argument 2 = 99
argument 3 = last_argument
>>>
```

U metodu plot() (u Programu 6-1) argumenti za analogne ulaze su nabrojani kao promenljive kanala i vrednosti y. Paleta promenljivih je lista različitih boja za svaki analogni ulaz.

6.2.2 ● Prosleđivanje argumenata funkciji pomoću argumenata ključnih reči

Ako pogledate metod __init__() za klasu, videćete da su dva argumenta argumenti ključnih reči, scale_x i scale_y. Oni se nalaze iza dva poziciona argumenta – visine i širine. Argumenti ključnih reči imaju podrazumevane vrednosti koje će biti upotrebljene ako se vrednosti ne proslede pozivanjem metoda. Te podrazumevane vrednosti se nalaze iza znakova jednakosti. Ako pozivalac prosledi vrednosti, argumenti ključnih reči koriste vrednosti koje su prosleđene.

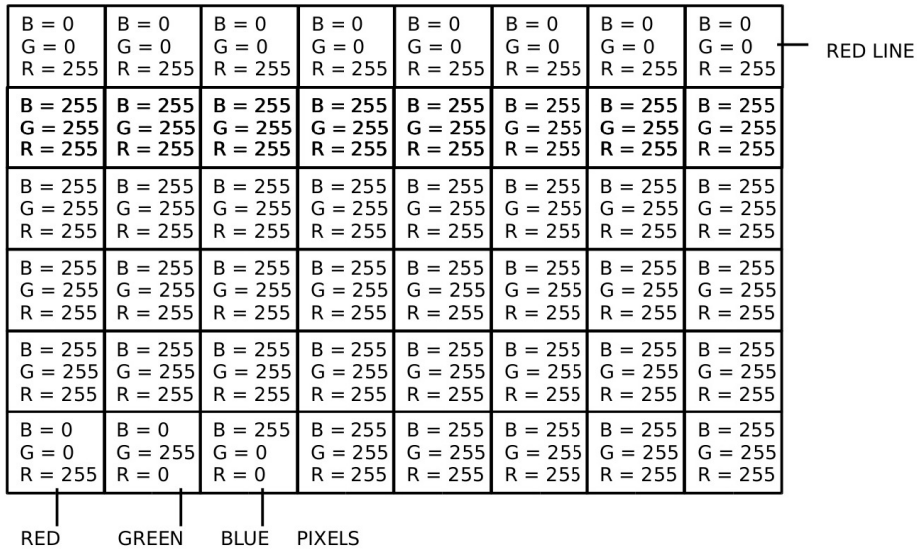
Poglavlje 7 • Grafikoni bitmape

7.1 • Uvod

U ovom poglavlju ćemo uglavnom razmatrati strukturu slika bitmape i kako da upravljate njima. U prethodnom poglavlju canvas vidžet se koristio za crtanje prilikom kreiranja GUI-ja u Tkinter-u. Šta ćete uraditi ako želite da nacrtate grafikon i sačuvate ga kao sliku datoteke? Python obezbeđuje metod za čuvanje podloga za crtanje kao PostScript datoteke, ali u ovom poglavlju ćemo kreirati i modifikovati slike bitmape koje se mogu sačuvati i čitati u bilo kom programu za prikazivanje slika. Slikama bitmape se može lako manipulisati kao da se nalaze u nizu piksela, kao tačke na komadu grafičkog papira.

7.2 • Datoteka bitmape

Za slike bitmape koje ćemo koristiti svaka boja piksela će biti definisana pomoću tri bajta – po jedan za svaku osnovnu boju. Na slici 7-7 je prikazana slika veličine 8 x 6 piksela. Postoji kvadrat za svaki piksel i bajt za svake tri boje. Slika je bela, osim crvene linije na vrhu i tri piksela počev od levog donjeg ugla koji su crveni, zeleni i plavi.

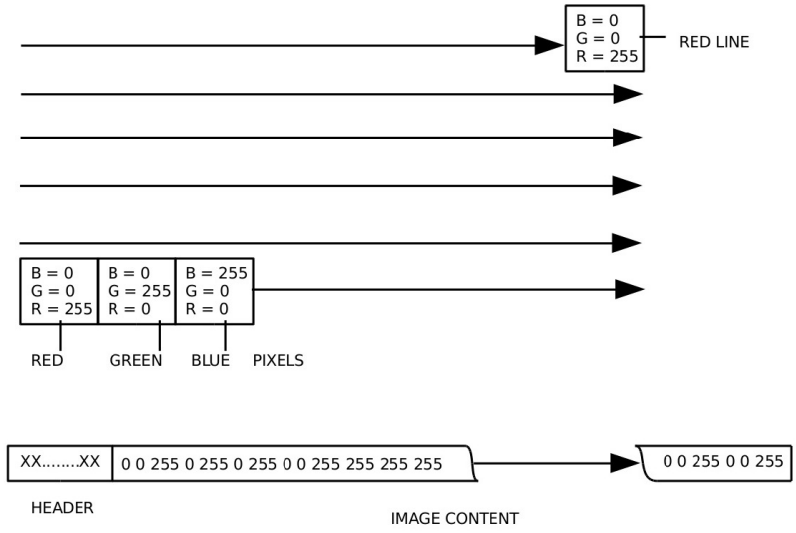


Slika 7-1 Slika veličine 8 x 6 piksela

Na slici 7-1 nije prikazano, ali datoteka bitmape počinje stringom od 54 bajtova, koji predstavljaju zaglavlje. Ono sadrži informacije o slici, ali sada nam nije važan sadržaj zaglavlja. Zatim sledi stvarni sadržaj slike, počev od donjeg levog ugla slike, a sledeća tri bajta predstavljaju vrednosti plavog, zelenog piksela i na kraju crvenog prvog piksela. Ovo se nastavlja za svaki piksel u donjem redu. Posle donjeg reda sledi levi najviši piksel sledećeg gornjeg reda i tako redom za ostatak slike. Da biste pregledali i modifikovali slike preporučujem KolourPaint. Ovaj program možete lako da instalirate na Debian-u tako što ćete uneti sledeće u komandnu liniju:

```
user@debian:~$ sudo apt-get install kolourpaint4
```

Slika koja je prikazana na slici 7-1 je u zip datoteci, čiji je naziv fig1image.bmp. Da biste prikazali sliku, moraćete maksimalno da je uvećate. Na slici 7-2 prikazano je kako su bajtovi koji definišu sliku sekvencirani u datoteci. Možete da vidite prva četiri piksela i poslednja dva piksela sa slike 7-1. Napominjemo da je raspored bajtova za svaki piksel sledeći: plavi piksel, zeleni i zatim crveni.



Slika 7-2 Sekvenca bajtova u datoteci

Cilj u ovoj vežbi je da napišemo datoteku slike pomoću našeg programa. Ova datoteka će biti izlazna datoteka. Iako nam sadržaj zaglavlja nije važan, zaglavlje je potrebno za svaku datoteku koju kreiramo. Na sreću zaglavlje ne zavisi od sadržaja, već samo od geometrije slike i dubine boje. Najlakši način za pisanje zaglavlja datoteke je da kopirate zaglavlje iz druge slike iste veličine i dubine boje. Kreiranje zaglavlja ćemo videti kasnije u ovom poglavlju.

Koristićemo postojeću bitmapiranu datoteku, tj. našu sliku šablona i pročitaćemo je. Zaglavlje će biti ponovo upotrebljeno kao zaglavlje za izlaznu datoteku. Datoteka šablona će programu omogućiti prednost u vidu pozadine, kao što su koordinatne linije ili strane instrumenta koje možete pratiti da biste crtali.

Ako ponovo pogledamo sliku 7-1, videćemo da je svaki kvadrat piksel i da je potrebno tri bajtova za postavljanje boje piksela. Možemo da kreiramo listu sa tri člana, u kojoj je svaki ceo broj od 0 do 255. To bi značilo da je treći piksel sa leve strane [255, 0, 0]. Ako imamo odvojenu sliku za svaki piksel, treba nam neki način na koji bi identifikovali gde se nalaze ove liste na slici. Python objekat koji je idealan za ovo je rečnik. Rečnike smo razmatrali u pasusu 2.6.5. Članovi rečnika su indeksirani pomoću ključa. Pogledajte ispod kako svaki piksel sa slike 7-1 može da se zadrži u rečniku. Ključevi koje ćemo koristiti da bismo indeksirali liste koje definišu boje piksela su n-torke sa dva cela broja. Ovi celi brojevi su horizontalni i vertikalni pomeraji sa donjeg levog ugla slike.

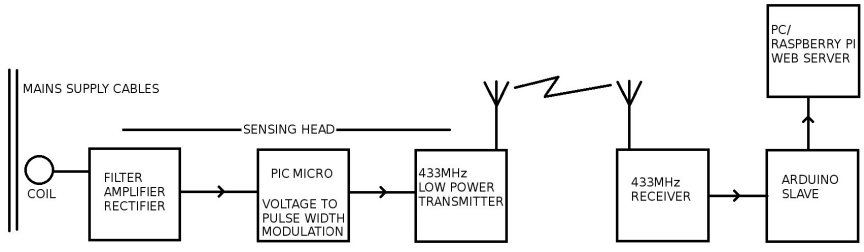
```
image_content_dictionary = {(7, 3): [255, 255, 255], (1, 3): [255, 255, 255], (3, 0):
[255, 255, 255],.............}
```

Poglavlje 8 • Projekat za praćenje potrošnje električne energije

8.1 • Uvod

U ovom projektu ćemo opisati veći projekat koji će pratiti potrošnju struje u naponskoj mreži. Prvo ćemo naglasiti da nije potrebna direktno povezivanje sa mrežom. Koristi se neinvazivna tehnika koja uključuje kalem žice, koji je postavljen blizu kablova za napajanje. Sonda će izmeriti struju koja teče i poslaće vrednost prijemniku koji će biti povezan sa Arduino slave uređajem. Biće iscrtan grafikon potrošnje struje tokom vremena. U sledećem poglavlju su prikazani detalji o tome kako se koristi veb pregledač kao GUI, koji može da se prikaže na mobilnom telefonu ili drugom računaru. Ovo će omogućiti da vidite rezultat kada niste kod kuće. Razlog za radio vezu je to što veoma često glavno brojilo u zgradi nije na pogodnom mestu za računar.

Daljinski senzor kojeg pokreće kalem se sastoji od filtera, pojačavača, preciznog polutalasnog ispravljača, PIC mikrokontrolera i radio predajnika male snage. PIC mikrokontroler je 12F1822 i može da se programira pomoću alatke koja je napisana u Python-u i koja koristi Arduino slave uređaj. Naveden je kod za PIC mikrokontroler. Predajnik i prijemnik se mogu nabaviti od kompanije Farnel Components, koja ima međunarodnu distribucijsku mrežu.



Slika 8-1 Blok dijagram praćenja mrežnog napajanja

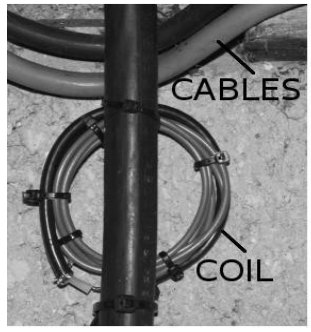
U ovom poglavlju ćemo predstaviti neke teme tako što ćemo iskoristiti prednosti desktop aplikacije koja je korišćena u ovom projektu.

8.2 • Sonda i daljinski senzor

Sonda koja je korišćena u prototipu je kalem izolovane žice, bez jezgra sa 5 namotaja i prečnikom od približno 60 mm, kao što je prikazano na slici 8-2. Kalem magneta je povezan sa senzorom pomoću dela koaksijalnog kabla. Može da se koristi kabl sa upredenim paricama. Ovo nije uobičajen način merenja struje u kablu za napajanje. Uobičajeno je da se koristi strujni transformator sa magnetnim jezgrom kroz koji prolazi jedan od kablova za napajanje. Jezgro ima namotaje koji su povezani sa kolom niske impedanse. Ovde su nedostaci to što strujni transformator mora da se poveže sa kablom pre nego što se priključi ili mora da se koristi transformator sa razdvojenim jezgrom. Ni jedna opcija nije pogodna za projekat izgradnje kuće, jer čak i ako koristite transformator sa podeljenim jezgrom, verovatno nećete imati mesta za njegovo priključivanje na kućnu instalaciju. Metod koji je opisan ovde je potpuno neinvazivan.

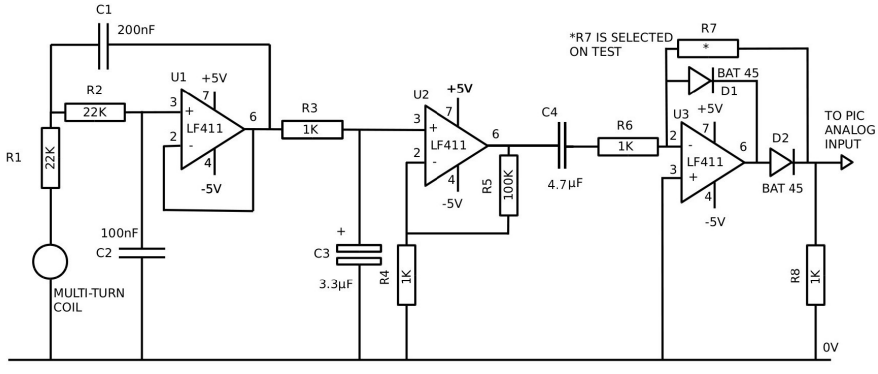
Način na koji ovaj senzorski kalem radi je tako što se neka magnetna polja iz dva kabla za napajanje povezuju sa kalemom. Pošto je jedan od kablova bliži kalemu, doći će do rezultujućeg polja koje indukuje napon u kalemu. Sada je ovaj način pogrešan. Trenutni magnetni fluks iz mrežnog napajanja je proporcionalan trenutnoj vrednosti struje u kablu za napajanje, ali napon koji je indukovan u kalemu je proporcionalan brzini promene magnetnog fluksa.

Ako uzmemo u obzir samo jednu frekvenciju (50 Hz u UK-a), maksimalna brzina promene fluksa je proporcionalna maksimalnoj trenutnoj vrednosti struje, a samo je faza pomerena za 90 stepeni. Međutim, prisutni su harmonici u struji i ako uporedite maksimalnu brzinu promene od 150 Hz sinusnog talasa sa maksimalnom brzinom promene od 50 Hz sinusnog talasa jednake amplitude, ona je pet puta veća za peti harmonik. Da bismo rešili ovaj problem, dodaje se niskopropusni filter u senzoru da bi se smanjio efekat bilo kog harmoničkog sadržaja u napajanju. Važno je napomenuti da ovo neće dati dovoljno precizne rezultate u svrhe merenja. Napominjemo, na slici možemo da vidimo dva kabla koja se nalaze iznad kalema. U UK-a se ovi kablovi nazivaju „meter tail" kablovi.



Slika 8-2 Prototipska sonda

Na slici 8-3 prikazana je električna šema dela senzora. Koriste se tri LF411 operaciona pojačavača. Prva faza „U1" se koristi kao Sallen-Key niskopropusni Batervotov filter sa ugaonom frekvencijom oko 50 Hz. Ako je vaše lokalno napajanje 60 Hz, ovo neće biti važno. Druga faza „U2" je neinvertujući predpojačavač sa dobitkom od 101. Još jedan niskopropusni filter je dodat tako što su korišćeni otpornik „R3" i kondenzator „C3". Poslednja faza „U3" je precizni polutalasni ispravljač. Dobitak poslednje faze je postavljen tako što je izabrana vrednost za otpornik „R7". Dobitak je odnos između otpornika „R7" i „R6". U mom prototipu vrednost od 80 k je korišćena za dobitak od 80. Možda vam je lakše da ovde koristite promenljivi otpornik. Njegova vrednost će biti određena prilikom testiranja. Dobro obratite pažnju da su prva dva operaciona pojačavača U1 i U2 povezana kao neinvertujući pojačavači, a treći U3 je povezan kao invertujući pojačavač. Drugim rečima, pinovi 2 i 3 će biti postavljeni kako treba. Kondenzator C4 treba da bude nepolarizovani tip, kao što je metalizirani poliester.



Slika 8-3 Električna šema senzora

```
            self.calibration_dict = {'raw_offset':0, 'raw_to_current':1, 'fullscale_current':100}

        self.circ_buffer = collections.deque([], maxlen=50)
        #add menus
        menubar = tk.Menu(self.master)
        master.config(menu=menubar)
        commissioning_menu = tk.Menu(menubar, tearoff = 0)
        menubar.add_cascade(label = 'Commissioning', menu = commissioning_menu)
        commissioning_menu.add_command(label = 'Calibration', command = self.calibration)
        commissioning_menu.add_command(label = 'Future Use')
        commissioning_menu.add_command(label = 'Future Use')
        commissioning_menu.add_separator()
        commissioning_menu.add_command(label = 'Future Use')
        screen_capture_menu = tk.Menu(menubar, tearoff = 0)
        menubar.add_cascade(label = 'Screen Capture', menu = screen_capture_menu)
        screen_capture_menu.add_command(label = 'Capture', command = self.capture)

        self.num_samples = self.X
        self.graph1 = plotter.Graph(master,self.X, self.Y,
        scale_x = self.num_samples, scale_y = self.calibration_dict['fullscale_current'])
        self.graph1.frm.grid(row = 0, column = 0, padx = 10, pady =10,columnspan = 3)
        self.reading()

    def reading(self):
        self.uno1.read_analog(1)
        raw = self.uno1.read_analog_dict[0]
        i = (raw-self.calibration_dict['raw_offset']) * self.calibration_dict['raw_to_current']
        y = int(i*self.Y/self.calibration_dict['fullscale_current'])
        self.circ_buffer.append([raw,i,y])
        if self.x%self.X == 0:
            self.graph1.flyback()

        self.graph1.plot(y)
        self.x = self.x + 1
        self.master.after(10000, self.reading)

    def calibration(self):
        self.cal_window = tk.Toplevel()
        self.cal_window.title('Calibration')
        self.cal_output = tk.Text(self.cal_window, width = 30, height = 20, relief = 'ridge')
        self.btn_save_cal = tk.Button(self.cal_window, text = 'Save Calibration', command = self.save_calibration)
        self.lbl_raw = tk.Label(self.cal_window, text = 'Raw/Current/Screen Position')
        self.lbl_offset = tk.Label(self.cal_window, text = 'raw_offset')
        self.strvar_offset = tk.StringVar()
        self.strvar_offset.set(self.calibration_dict['raw_offset'])
        self.ent_offset = tk.Entry(self.cal_window, textvariable = self.strvar_offset, width = 10)
        self.lbl_raw_to_current = tk.Label(self.cal_window, text = 'raw_to_current')
        self.strvar_raw_to_current = tk.StringVar()
        self.strvar_raw_to_current.set(self.calibration_dict['raw_to_current'])
        self.ent_raw_to_current = tk.Entry(self.cal_window, textvariable = self.strvar_raw_to_current, width = 10)
        self.lbl_fullscale_current = tk.Label(self.cal_window, text = 'fullscale_current')
        self.strvar_fullscale_current = tk.StringVar()
        self.strvar_fullscale_current.set(self.calibration_dict['fullscale_current'])
        self.ent_fullscale_current = tk.Entry(self.cal_window, textvariable = self.strvar_fullscale_current,
            width = 10)
        self.cal_window.grid()
        self.lbl_raw.grid(row = 0, column = 0)
        self.cal_output.grid(row = 1, column = 0, rowspan  = 7)
        self.lbl_offset.grid(row = 1, column = 1, sticky = 'S')
        self.ent_offset.grid(row = 2, column = 1, sticky = 'N')
        self.lbl_raw_to_current.grid(row = 3, column = 1, sticky = tk.S)
        self.ent_raw_to_current.grid(row = 4, column = 1,  sticky = tk.N)
        self.lbl_fullscale_current.grid(row = 5, column = 1, sticky = tk.S)
```

```
        self.ent_fullscale_current.grid(row = 6, column = 1, sticky = tk.N)
        self.btn_save_cal.grid(row = 7, column = 1, padx = 5, pady = 5, sticky = tk.S)
        self.calibration_update()

    def calibration_update(self):
        try:
            self.cal_output.delete(1.0, tk.END)
            for data in self.circ_buffer:
                self.cal_output.insert(tk.END,'raw = {0:d} i = {1:0.2f}A y = {2:d}\n'.format(data[0], data[1],
                    data[2]))
                self.cal_output.see(tk.END)
        except:
            self.master.after_cancel(self.after_handle)

        self.after_handle =  self.master.after(10000, self.calibration_update)

    def save_calibration(self):
        self.calibration_dict['raw_offset'] = float(self.strvar_offset.get())
        self.calibration_dict['raw_to_current'] = float(self.strvar_raw_to_current.get())
        self.calibration_dict['fullscale_current']= float(self.strvar_fullscale_current.get())
        try:
            calibration_file = open('current_monitor_cal.dat','bw')
            pickle.dump(self.calibration_dict, calibration_file)
            calibration_file.close()
        except:
            self.cal_output.insert(END,'\vFailed to open file to save calibration\n')
            self.cal_output.see(END)

    def capture(self):
        file_name = tk.filedialog.asksaveasfilename(filetypes = [('Bit Map','*.bmp')],title = 'Choose or Enter
            File Name')
        if file_name:
            segments = []
            tags = self.graph1.can.find_all()                       # find all objects on the canvas
            for tag in tags:                                        # loop through objects
                if self.graph1.can.type(tag) == 'line':             # filter out the lines
                    segments.append(self.graph1.can.coords(tag))    # make a list of coordinate lists

            draw1 = bmp_draw.BmpDraw('current_monitor_screen_cap_template.bmpt')
                if draw1.flag == True:
                    for x in range (0,self.X, 90):
                        draw1.line(x, 0, x, self.Y, [0,255,0])

                    for y in range (0, self.Y, 50 ):
                        draw1.line(0, y, self.X, y, [0,255,0])

                    for segment in segments:
                        draw1.line(segment[0], self.Y - segment[1], segment[2], self.Y - segment[3], [0,0,255])

                draw1.write_file(file_name)
                tk.messagebox.showinfo('Capture', 'Capture Saved')
        else:
            tk.messagebox.showinfo("Capture", 'Capture Cancelled')

tkin = tk.Tk()
front_end = CurrentLogger(tkin)
tkin.mainloop()
```

Poglavlje 9 • Interfejsi veb pregledača

9.1 • Uvod

Desktop GUI aplikacije su sasvim dovoljne ako želite da se povežete sa nekim lokalnim hardverom. Međutim, često je pogodnije da koristite mobilni uređaj, kao što je tablični računar ili mobilni telefon. To će vam omogućiti da pristupite vašim hardverskim uređajima daljinskim putem, čak i kada niste kod kuće. Treba da razmotrite bezbednosne mere kada daljinski kontrolišete stvarne električne i elektromehaničke uređaje. U ovom poglavlju ćemo razmotriti praktičnu upotrebu GUI-a za vaše projekte, pa svakako da ovo poglavlje ne sadrži informacije o veb dizajnu. U ovom poglavlju ćemo se nadovezati na projekat praćenja potrošnje struje iz prethodnog poglavlja. Možete lako da primenite informacije iz ovog poglavlja na druge projekte koje osmislite i u kojima ćete možda želeti da pratite slične analogne ili digitalne ulaze na Arduino slave uređaju.

9.2 • HTML 5

Da bismo generisali grafički interfejs, koristićemo HTML5 za prikaz grafikona iz projekta praćenja potrošnje struje u prethodnom poglavlju. HTML označava jezik za pravljenje hiperteksta (engl. Hypertext Markup Language). On definiše veb stranicu u elementima, koji su predstavljeni pomoću tagova. Ovde ćemo razmatrati samo ono najosnovnije. Treba samo da generišemo i prikažemo sliku koje je slična Python canvas vidžetu u desktop programu. Iako HTML 5 ima objekat canvas, nećemo ga koristiti, već ćemo koristiti Scaleable Vector Graphics (SVG). Kod koji treba da napišemo je veoma jednostavan.

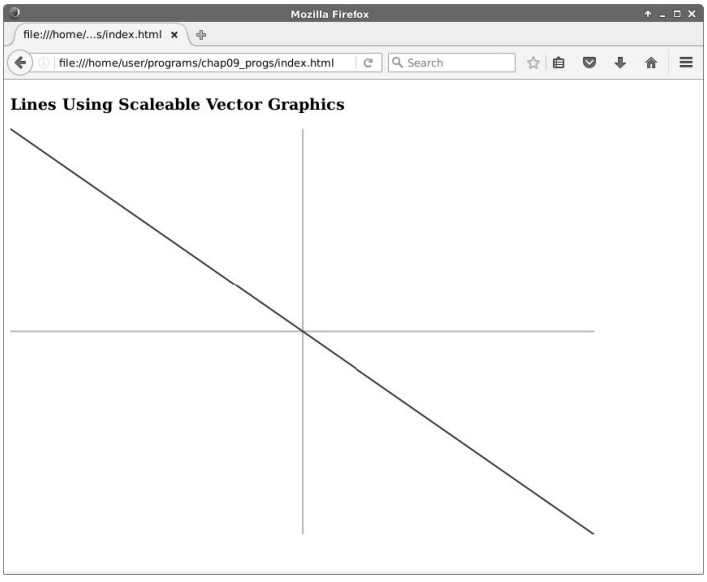
Program 9-1 Index.html

```
<!--prog_09_01 is actually file index.html in directory ~/programs/chap09-->
<!DOCTYPE html>
<html>
<body>
<h3>Lines Using Scaleable Vector Graphics</h3>
<svg height=”500” width=”720”>
    <line x1=”360” y1=”0” x2=”360” y2=”500” style=”stroke:rgb(0,255,0);stroke-width:2” /><!-- vertical line -->
    <line x1=”0” y1=”250” x2=”720” y2=”250” style=”stroke:rgb(0,255,0);stroke-width:2” /><!-- horizontal line -->
    <line x1=”0” y1=”0” x2=”720” y2=”500” style=”stroke:rgb(255,0,0);stroke-width:2” /><!-- diagonal -->
</svg>
</body></html>
```

Gornji HTML skript možete da prikažete u veb pregledaču tako što ćete ukucati /home/ u adresnoj traci i zatim pristupiti datoteci koja se nalazi ispod programa za Poglavlje 9. Naziv datoteke je index.html. Prva linija je komentar i nema nikakav uticaj na veb stranicu. Komentari su zatvoreni u tagovima “<!—” i “—>”. Sledeća linija ukazuje pregledaču na tip dokumenta. Prvi tag je “<html>” i ima završni tag na poslednjoj liniji “</html>”. Sledeći par tagova obuhvata zaglavlje, a zatim sledi SVG kod koji je zatvoren u svg tagovima. Početni tagovi definišu veličinu grafikona.

Linije koje su iscrtane se definišu između tagova “<line” i “/>”. Linije se definišu na sličan način koji smo primenili na Python podlogu za crtanje – koristeći x i y koordinate za svaki kraj linije, boju definisanu crvenim, zelenim i plavim vrednostima i vrednost širine poteza. Na slici 9-1 prikazan je dobijeni grafikon koji je lokalno prikazan pomoću putanje datoteke. Da biste prikazali veb stranicu na drugačijem uređaju od onog u kojem se nalazi datoteka, potreban je veb server. U prvom izdanju

ove knjige korišćeni su Apache Web Server i skriptovi opšteg programskog interfejsa (CGI-ja) za generisanje interaktivnih veb stranica. U ovom izdanju koristićemo CherryPy Web radni okvir, koji omogućava mnogo jednostavniji način za kreiranje naših dinamičkih veb stranica.



Slika 9-1

9.3 • CherryPy Python veb radni okvir

CherryPy je veb radni okvir za Python koji nam obezbeđuje veb server i okruženje za opsluživanje dinamičkog sadržaja koji omogućavaju korisniku da komunicira sa Python programima koji su pokrenuti na serveru. Da biste instalirali CherryPy, treba da ukucate sledeće u komandnu liniju:

```
user@debian:~$ sudo apt-get install python3-cherrypy3
```

Počećemo od sadržaja datoteke „index.html", koja je prikazana iznad, i opslužiti je koristeći radni okvir CherryPy.

Program 9-2 CherryPy

```
#!/usr/bin/env python3
#prog_09_02.py
import cherrypy

class SvgDemo:
    @cherrypy.expose
    def index(self):
        return '''
<!DOCTYPE html>
<html>
<body>
<h3>Lines Using Scaleable Vector Graphics</h3>
<svg height="500" width="720">
<line x1="360" y1="0" x2="360" y2="500" style="stroke:rgb(0,255,0);stroke-width:2" /><!-- vertical line -->
```

Poglavlje 10 • Curses interfejsi

10.1 • Uvod

Curses interfejsi rade na terminalu. Curses je Python paket za biblioteku ncurses koja vam omogućava da pišete korisničke interfejse zasnovane na tekstu, koji ne zavise od terminala koji se koristi. Iako se interfejs zasniva na znakovima, mogu da se kreiraju mnoge funkcije GUI-ja. Jedan primer ovog tipa je Midnight Commander File Manager. Za razliku od osnovnog programa komandne linije, interfejs curses omogućava da ispisujete bilo gde na ekranu i ima više prozora za unos i čitanje podataka. Neke ograničene grafike, kao što su trake koje menjaju njihove dužine, mogu da se koriste kao indikatori merenja i takođe prozori koji menjaju boju mogu da se koriste kao indikatori. Razlog zbog kojeg želite da ih koristite je to što mogu da rade bez X prozora i zato su korisni na računarima malih resursa. Curses je već deo Python-a i lakše se koristi od biblioteke ncurses, koju biste koristili ako programirate na jeziku C. Curses paket nije dostupan za Python na Microsoft Windows-u. Na slici 10-1 prikazan je primer merača. Ne postoje gotovi vidžeti, ali možete jednostavno da napravite svoje vidžete kao klase, koje možete ponovo da upotrebite.



Slika 10-1 Prikaz merača

10.2 • Početak rada sa meračima

Za početak primer je Program 10 -1 koji ispisuje poruku na ekranu na dve različite pozicije. Uobičajeno da se poruka ispiše na poziciji kursora i ne možete da birate proizvoljnu poziciju na ekranu, ali pomoću Curses to možete.

Program 10-1 Ispisivanje poruke na ekranu

```
#!/usr/bin/env python3
#prog_10_01.py

import curses

stdscr = curses.initscr()
curses.noecho()
curses.cbreak()
stdscr.addstr(5, 20, 'Curses can write anywhere on the screen')
stdscr.addstr(7, 10, 'Curses can write anywhere on the screen')
stdscr.getch()

curses.echo()
curses.nocbreak()
curses.endwin()
```

Na slici 10-2 prikazan je izlaz ovog programa. Prvo morate da uvezete curses i zatim da pozovete funkciju curses.initscr(). Funkcija inicijalizuje biblioteku i vraća objekat prozora koji uobičajeno poziva stdscr, koji predstavlja ceo ekran. U stdscr-u može da se napiše tekst i mogu da se postave zasebni prozori. Ovde nema prozora, samo stdscr. Funkcija curses.noecho() onemogućava pojavu znakova dok kucate: postoje posebne funkcije za prikaz znakova. Funkcija curses.cbreak() zaustavlja privremeno

skladištenje ulaza u memoriju, a u običnom terminalu morate da pritisnite taster return. Da bi tekst bio ispisan na ekranu, poziva se funkcija addstr().

Ovo je metod ekrana ili prozora i može da koristi argumente za pozicioniranje teksta, string za ispis i takođe atribute za tekst, ali ovde ih nema. Važno je da obratite pažnju na redosled pozicijskih argumenata. Prvi argument je broj linija na ekranu ili u prozoru, a drugi je broj kolona ili mesta znakova duž ekrana sa leve strane. Curses ne iscrtava pojedinačne piksele, samo znakove.



Slika 10-2 Izlaz programa

Da biste sprečili da se program završi pre nego što to vidite, funkcija stdscr.getch() čeka pritisak na neki taster. Pre nego se program završi, morate da poništite promene koje ste načinili u operaciji terminala, u suprotnom terminal se neće ponašati uobičajeno nakon što se program završi. U programu možete da vidite tri funkcije koje se pozivaju radi inverzije ovih efekata. Ako program ne uspe pre nego što se pozovu ove funkcije, terminal će biti u neupotrebljivom stanju koje će zahtevati da se zatvori jedan terminal i otvori drugi. Međutim, postoji prikladniji način za rešavanje ovog problema.

10.2.1 • Funkcija wrapper

Funkcija wrapper obrađuje ovaj problem tako što morate ručno pozvati ove funkcije. Morate da kod stavite u funkciju, a u ovom slučaju funkcija se naziva main() i mora da ima jedan argument pod nazivom stdscr. Program se pokreće tako što poziva funkciju curses.wrapper() kojoj mora da se dodeli naziv vaše funkcije pre nego što je pozove. Kada funkcija wrapper pozove vašu funkciju, prosleđuje je objektu stdscr. Kada se funkcija završi, program vraća funkciju wrapper i briše terminal. Ako funkcija ne uspe iz bilo kog razloga, funkcija wrapper će upotrebiti rutinu try/exception koja upravlja brisanjem.

Program 10-2 Upotreba funkcije wrapper

```
#!/usr/bin/env python3
#prog_10_02.py
import curses

def main(stdscr):
    stdscr.addstr(5, 20, 'Curses can write anywhere on the screen')
    stdscr.addstr(7, 10, 'Curses can write anywhere on the screen')
    stdscr.getch()

curses.wrapper(main)
```

Dodatak A • Izvorni kod za Arduino Uno Slave

Program A-1 je izvorni kod za Arduino slave i nije Python već Arduino dijalekt jezika C i C++. Ovaj izvorni kod postaje komplikovan za hex datoteku koja je izvršivi kod, koji se učitava u Arduino kako bi postao slave program. Datoteka hex je u zip datoteci kao Arduino_Slave_17_03_26a.ino.standard.hex. Objašnjenje ovog programa nije dato, ali ako ste upoznati sa jezikom C, pogledajte sliku 5-12, koja predstavlja dijagram stanja za ovaj program.

Program A-1 Arduino skica

```
/*Arduino_Slave_17_03_26a.ino*/
#include <EEPROM.h>

#define STATE_IDLE                0
#define STATE_CHECK_TELEGRAM      1
#define STATE_CONFIGURE_IO        2
#define STATE_DIG_READ            3
#define STATE_DIG_WRITE           4
#define STATE_AN_READ             5
#define STATE_AN_WRITE            6

#define STATE_FAIL_SAFE           7
#define STATE_TX_IDS              8
#define STATE_SET_IDS             9
#define STATE_RECEIVING_BYTES     10

#define FUN_CODE_CONFIG           2
#define FUN_CODE_DIG_READ         3
#define FUN_CODE_DIG_WRITE        4
#define FUN_CODE_AN_READ          5
#define FUN_CODE_AN_WRITE         6
#define FUN_CODE_TX_IDS           8
#define FUN_CODE_SET_IDS          9

#define BROADCAST_ADDR    255

#define ERROR_CHECK_SUM           1
#define ERROR_FUN_CODE            2
#define ERROR_ADDRESS             3
#define ERROR_PROJECT_ID          4

#define SLAVE_ADDRESS_EEPROM_LOCATION  0
#define PROJECT_ID_H_EEPROM_LOCATION   1
#define PROJECT_ID_L_EEPROM_LOCATION   2

unsigned char request_telegram_ascii[64];
unsigned char request_telegram_binary[32];
unsigned char response_telegram_ascii[64];
unsigned char response_telegram_binary[32];
unsigned char telegram_index = 0;
unsigned char telegram_binary_length;
unsigned char slave_address;
unsigned char project_id_h;
unsigned char project_id_l;
unsigned char state = 0;
volatile unsigned char chrbuff;
unsigned char pullupmask_d;
unsigned char pullupmask_b;
```

Dodatak B – Python Uno Slave

U nastavku je dat program uno_slave.py. Ovo je komunikacioni program koji „komanduje" Arduino slave programom.

Program B-1 uno_slave.py

```
#!/usr/bin/python3
import serial
import os

H = 1
L = 0
FUN_CODE_DIG_READ = 3
FUN_CODE_DIG_WRITE = 4
FUN_CODE_AN_READ = 5
FUN_CODE_AN_WRITE = 6
FUN_CODE_GET_IDS = 8 #maps to FUN_CODE_TX_IDs in the arduino code
FUN_CODE_CONFIG = 2
BROADCAST_ADDR = 255

DEBUG = False

class Uno:
    '''Arduino Uno as an IO slave.

    The Uno must be running the Arduino_slavexxx sketch.
    Uses an ascii formatted communications telegram.

    '''

    def __init__(self,address = 0, project = 0, config_digital = []):
        '''Initialise the Uno slave.

        address -- 1 - 254.
        project -- 0 - 65535
        config_digital -- List of integers detailing the pins configured as outputs.

        '''

        self.project = project
        self.address = address
        self.config_digital = config_digital
        self.port = None
        self.rw_available__flag = False
        self.ser = serial.Serial()
        self.ser.timeout = 0.1
        self.ser.bytesize = 8
        self.ser.stopbits = 2
        self.ser.baudrate = 9600
        self.ser.xonxoff = False
        self.write_digital_dict = {}
        self.write_analog_dict = {}
        self.read_digital_dict = {0:0, 1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0, 11:0, 12:0, 13:0, }
        self.read_analog_dict = {0:0, 1:0, 2:0, 3:0, 4:0, 5:0}

    def connection(self):
        '''This function checks for connection and digital configuration and calls connect() if necessary.

        Is called by read_digital(), read_analog(), write_digital(), and write_analog() to check for connection to slave.
        Return True or False.

        '''
```