NAVODILA ZA UPORABO

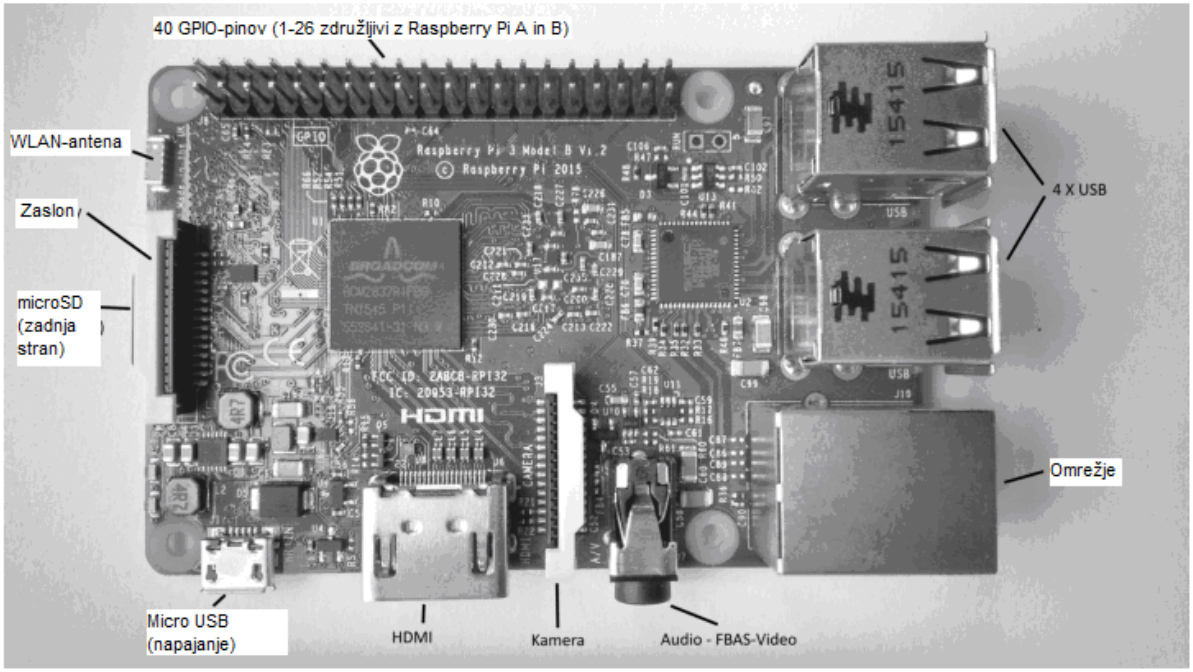
Učni komplet za Raspberry Pi Conrad Components

Kataloška št.: **12 25 953**

Kazalo

1 Od namestitve operacijskega sistema do prvega programa Python

O malokateri elektronski napravi v njenem cenovnem razredu se je v zadnjih letih toliko govorilo kot o Raspberry Pi. Tudi če na prvi pogled sploh ni videti, je Raspberry Pi polno funkcionalen računalnik po zelo ugodni ceni, ki je približno tako velik kot kreditna kartica. Ugodna ni samo strojna oprema, temveč tudi programska oprema: Operacijski sistem in vse aplikacije, ki jih potrebujete v vsakdanjiku, si lahko brezplačno prenesete s spleta.



Slika 1.1: Raspberry Pi in njegovi priključki za strojno opremo

S posebej prilagojenim operacijskim sistemom Linux z grafičnim vmesnikom je Raspberry Pi energijsko varčen, neslišen nadomestek za osebni računalnik. Njegov poljubno programirljiv GPIO-vmesnik poskrbi, da je Raspberry Pi posebej zanimiv za izdelovalce strojne opreme in sceno izdelovalcev oz. makerjev.

1.1 Kaj potrebujete?

Raspberry Pi je kljub svoji zelo majhni velikosti polno funkcionalen računalnik. Da ga lahko uporabljate, potrebujete še nekaj opreme tako kot pri "običajnem" računalniku: operacijski sistem, napajalnik, omrežje, monitor, tipkovnico in različne priključne kable.

1.1.1 Polnilnik za mobilni telefon Micro USB

Za Raspberry Pi zadostuje vsak sodoben polnilnik za mobilni telefon. Starejši polnilniki iz začetkov USB polnilne tehnike so še prešibki. Ko priključite USB-naprave z večjo porabo, kot

so zunanji trdi diski brez lastnega napajanja, potem potrebujete močnejši napajalnik. Napajalnik mora nuditi 5 V in minimalno 2.000 mA. Vgrajen regulator moči prepreči, da bi pri premočnih napajalnikih naprave "pregorele".

Pokazatelj prešibkega napajalnika
Ko se Raspberry Pi sicer zažene, vendar pa nato ne morete premikati miškinega kazalca ali pa se sistem ne odziva na vnose prek tipkovnice, to pomeni, da je na voljo prešibko napajanje. Tudi ko ne morete dostopati do priključenih USB-ključev ali trdih diskov, morate uporabiti močnejši napajalnik.

1.1.2 Spominska kartica

Spominska kartica ima pri Raspberry Pi vlogo trdega diska. Vsebuje operacijski sistem. Lastni podatki in nameščeni programi se prav tako shranjujejo nanjo. Spominska kartica mora biti velika najmanj 4 GB in mora po podatkih proizvajalca podpirati najmanj standard Class 4 (hitrostni razred 4). Ta standard navaja hitrost spominske kartice. Trenutna spominska kartica razreda 10 poskrbi za občutno boljšo zmogljivost. Vsi trenutni modeli Raspberry Pi uporabljajo spominske kartice microSD, ki jih poznamo tudi pri pametnih telefonih.

1.1.3 Tipkovnica

Uporabite lahko vsako običajno tipkovnico z USB-priključkom. Brezžične tipkovnice včasih ne delujejo, saj potrebujejo preveč toka ali posebne gonilnike. Če nimate na voljo nobene druge tipkovnice, potem za napajanje brezžične tipkovnice potrebujete USB-razdelilnik z ločenim napajanjem.

1.1.4 Miška

Miško z USB-priključkom potrebujete samo takrat, ko na Raspberry Pi uporabljate operacijski sistem z grafičnim uporabniškim vmesnikom, ter tudi za preizkuse v tem učnem kompletu.

1.1.5 Omrežni kabel

Za povezavo z usmerjevalnikom v lokalnem omrežju potrebujete omrežni kabel. Raspberry Pi 3 ima v ta namen tudi vgrajen WLAN-modul. Brez internetnega dostopa številnih funkcij računalnika Raspberry Pi ni možno smiselno uporabljati.

1.1.6 HDMI-kabel

Raspberry Pi lahko prek HDMI-kabla priključite na monitorje ali televizorje. Za priključitev na računalniške monitorje z DVI-priključkom obstajajo posebni HDMI-kabli ali adapterji.

1.1.7 Avdio kabel

Prek avdio kabla s 3,5 mm banana vtiči lahko v kombinaciji z Raspberry Pi uporabljate slušalke ali računalniške zvočnike. Avdio signal je na voljo tudi prek HDMI-kabla. Pri HDMI-televizorjih ali monitorjih ne potrebujete avdio kabla. Ko priključite računalniški monitor prek HDMI-kabla z DVI-adapterjem, potem se na tem mestu avdio signal ponavadi izgubi, tako da spet potrebujete analogni avdio izhod.

1.1.8 Rumeni video kabel FBAS

Če nimate na voljo HDMI-monitorja, potem lahko Raspberry Pi z analognim video kablom priključite na klasični televizor, vendar je pri tem ločljivost zaslona zelo majhna. Analogni

video izhod je kombiniran z avdio izhodom. Za priključitev na značilni rumeni priključek televizorja potrebujete adapterski kabel s 3,5 mm banana vtičem. Za televizorje brez rumenega FBAS-vhoda so na voljo adapterji iz FBAS na SCART. Pri analogni ločljivosti televizorja imate omejeno možnost upravljanja grafičnega vmesnika.

1.2 Namestitev operacijskega sistema Raspbian

Raspberry Pi prejmete brez operacijskega sistema. Za razliko od osebnih računalnikov, ki skoraj vsi uporabljajo operacijski sistem Windows, se za Raspberry Pi priporoča posebej prilagojen operacijski sistem Linux. Operacijski sistem Windows na varčni strojni opremi sploh ne bi deloval.

Raspbian je distribucija Linux, ki jo priporoča in podpira proizvajalec računalnika Raspberry Pi. Raspbian temelji na Debian-Linux, ki je ena izmed najbolj znanih distribucij Linux in je med drugim osnova za priljubljeni različici operacijskega sistema Linux kot sta Ubuntu in Knoppix. Kar je pri osebnih računalnikih trdi disk, je pri Raspberry Pi spominska kartica. Na njej se nahaja operacijski sistem in podatki. S te spominske kartice se Raspberry Pi tudi zaganja.

1.2.1 Priprava spominske kartice v računalniku

Ker se Raspberry Pi sam še ne more zagnati, je treba spominsko kartico pripraviti na osebnem računalniku. Pri tem na osebnem računalniku potrebujete čitalnik kartic. Lahko je fiksno vgrajen ali pa ga priključite prek USB-priključka.

Najbolje je, da uporabite nove spominske kartice, saj jih je proizvajalec že optimalno predhodno formatiral. Uporabite pa lahko tudi spominsko kartico, ki ste jo prej že uporabljali v digitalnem fotoaparatu ali pametnem telefonu. Te spominske kartice je treba pred uporabo za Raspberry Pi na novo formatirati. Teoretično lahko pri tem uporabite funkcije za formatiranje operacijskega sistema Windows. Občutno boljša je programska oprema »SDFormatter« ponudnika SD Association. Z njo formatirate spominske kartice za optimalno zmogljivost. To orodje si lahko brezplačno prenesete s strani www.sdcard.org/downloads/formatter_4.

Če ima spominska kartica particije iz prejšnje namestitve operacijskega sistema za Raspberry Pi, potem program SDFormatter ne prikazuje celotne velikosti. V tem primeru uporabite možnost formatiranja *FULL (Erase) (POLNO (Izbriši))* in aktivirajte možnost *Format Size Adjustment (Nastavitev velikosti formatiranja)*. S tem se na novo ustvarijo particije spominske kartice.

Podatki na spominski kartici se izbrišejo
Najbolje je, da za namestitev operacijskega sistema uporabite prazno spominsko kartico. Če se na spominski kartici nahajajo podatki, potem se ti pri ponovnem formatiranju med namestitvijo operacijskega sistema nepreklicno izbrišejo.

1.2.2 Namestitveni program NOOBS

»New Out Of Box Software« (NOOBS) je posebej enostaven namestitveni program za operacijske sisteme Raspberry Pi. Pri tem se za nastavitev zagonske spominske kartice uporabniku ni več treba ukvarjati s slikovnimi orodji in zagonskimi bloki, tako kot je to veljalo do nedavnega. NOOBS nudi možnost izbire med različnimi operacijskimi sistemi, pri čemer lahko pri prvem zagonu neposredno na Raspberry Pi izberete želeni operacijski sistem, ki se nato namesti na spominsko kartico z možnostjo zagona. Iz uradne spletne strani za prenose www.raspberrypi.org/downloads si prenesite namestitveno arhivsko datoteko NOOBS v velikosti pribl. 1,2 GB in jo na računalniku ekstrahirajte na spominsko kartico, ki je velika

najmanj 4 GB. Preizkusi v tem učnem kompletu so bili testirani s programom NOOBS v različici 1.9.2. Starejše različice so samo omejeno združljive z aktualnimi modeli Raspberry Pi.

Sedaj s to spominsko kartico zaženite Raspberry Pi. Pri tem jo vstavite v režo na Raspberry Pi in priključite tipkovnico, miško, monitor in omrežni kabel. Električni USB-priključek je na vrsti nazadnje. Prek njega se Raspberry Pi vključi. Ločena tipka za vklop ni na voljo.

Čez nekaj sekund se pojavi izbirni meni, v katerem lahko izberete želeni operacijski sistem. Mi uporabljamo operacijski sistem Raspbian, ki ga priporoča Združenje Raspberry Pi.

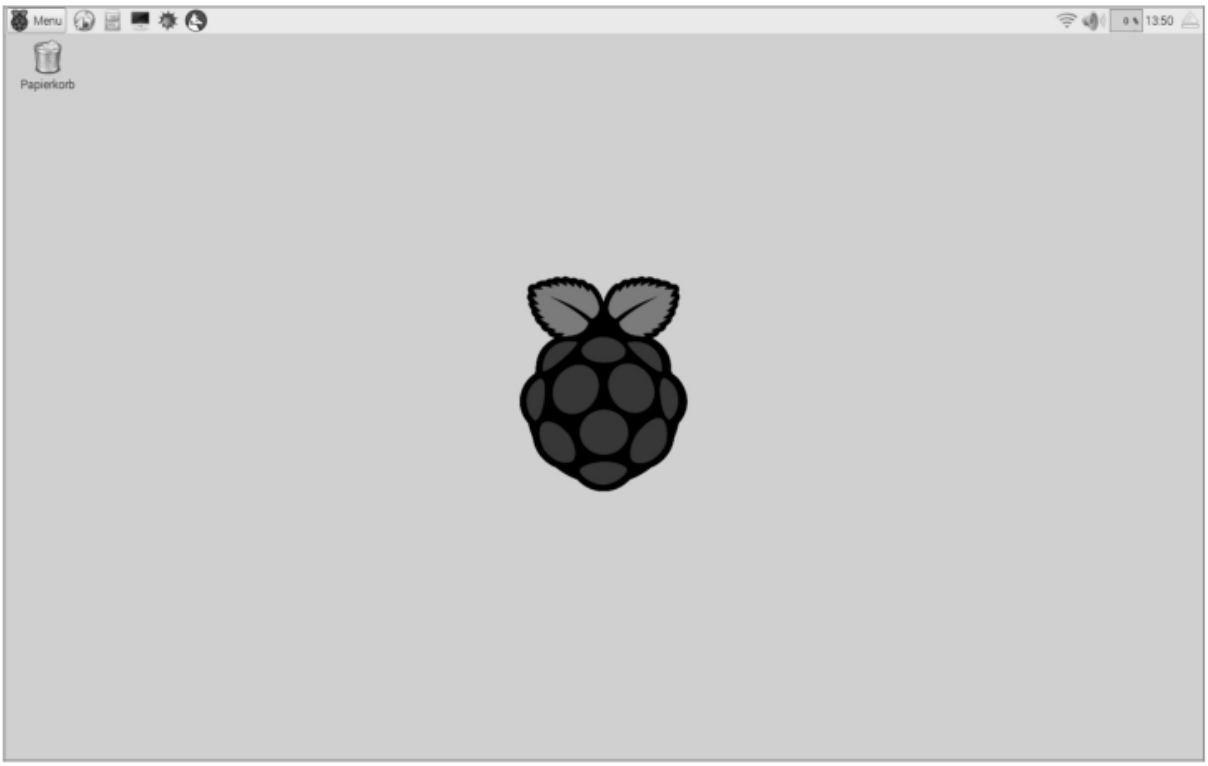
Čisto spodaj kot namestitveni jezik izberite nemščino (Deutsch) in označite predhodno izbran operacijski sistem Raspbian. Potem ko potrdite varnostno sporočilo, da se bo spominska kartica prepisala, se začne namestitev, ki traja nekaj minut. Med namestitvijo so prikazane kratke informacije o operacijskem sistemu Raspbian.

1.2.3 Priv začetek na Raspberry Pi

Po zaključeni namestitvi se Raspberry Pi ponovno zažene in samodejno aktivira grafično namizje LXDE. Nemški jezik in nemška razporeditev tipkovnice sta tako kot ostale pomembne osnovne nastavitve že samodejno izbrana. Po ponovnem zagonu je na voljo grafično namizje LXDE.

1.3 Skoraj tak kot Windows – grafični vmesnik LXDE

Številni se pri besedi »Linux« najprej zgrozijo, saj se bojijo, da bodo morali vnašati kriptična zaporedja ukazov prek ukazne vrstice tako kot pred 30 leti v operacijskem sistemu DOS. Daleč od tega! Linux kot odprti operacijski sistem razvijalcem nudi vse možnosti za razvoj lastnih grafičnih vmesnikov. Tako kot uporabnik operacijskega sistema, ki je v osnovi še vedno orientiran na ukazne vrstice, niste omejeni na en vmesnik.



Slika 1.2: Namizje LXDE na Raspberry Pi

Raspbian Linux za Raspberry Pi uporablja vmesnik LXDE (Lightweight X11 Desktop Environment), ki po eni strani potrebuje zelo malo sistemskih virov, po drugi strani pa je s svojim menijem Start in upraviteljem datotek zelo podoben znanemu vmesniku Windows – s to razliko, da se opravilna vrstica nahaja na zgornjem robu zaslona.

Prijava v Linux
Celo prijava uporabnika, ki je značilna za Linux, se opravi v ozadju. Če pa boste te podatke kljub temu kdaj potrebovali: Uporabniško ime se glasi `pi`, geslo pa `raspberry`.

Simbol levo zgoraj odpre meni Start, simbola zraven pa spletni brskalnik in upravitelja datotek. Meni Start je tako kot pri Windows sestavljen iz več stopenj. Programe, ki jih pogosto uporabljate, lahko z desnim klikom shranite na namizje.

Izklop računalnika Raspberry Pi
V teoriji lahko pri Raspberry Pi enostavno izvlečete električni vtič iz električne vtičnice in naprava se izključi. Vendar je bolje, da sistem pravilno zaustavite tako kot pri osebnem računalniku. Pri tem v meniju izberite možnost Shutdown (Zaustavitev).

1.3.1 Nastavitev časa

Raspberry Pi nima interne ure z dejanskim časom, temveč podatek o aktualnem času pridobi iz interneta. Vendar pa je tudi pri aktivni internetni povezavi na uri zgoraj desno v opravilni vrstici najprej prikazan napačen čas. Vzrok za to je standardno nastavljen časovni pas.

Izberite točko *Einstellungen/Raspberry Pi Configuration (Nastavitve/Konfiguracija Raspberry Pi)*. To pogovorno okno nadomesti konfiguracijsko orodje na osnovi besedila prejšnjih različic Raspbian. Na zavihku *Localisation (Lokalizacija)* kliknite na gumb *Set Timezone (Nastavi časovni pas)* in izberite časovni Europe/Berlin, ki se uporablja pri nas.

Slika 1.3: Nastavitev časovnega pasu

Če nimate internetne povezave, Raspberry Pi prikazuje neveljaven časovni pas. V tem primeru lahko prek simbola s črnim zaslonom v opravilni vrstici odprete okno z ukazno vrstico in nastavite pravilen čas, npr.:

```
sudo date -s "2016-03-22 08:00:00 CET"
```

Nastavitev se nato potrdi z besedilnim prikazom datuma in časa:

```
Di 22. Mär 08:00:00 CET 2016
```

Ta nastavitev velja samo do naslednjega ponovnega zagona. Ura z rezervnim baterijskim napajanjem ni na voljo.

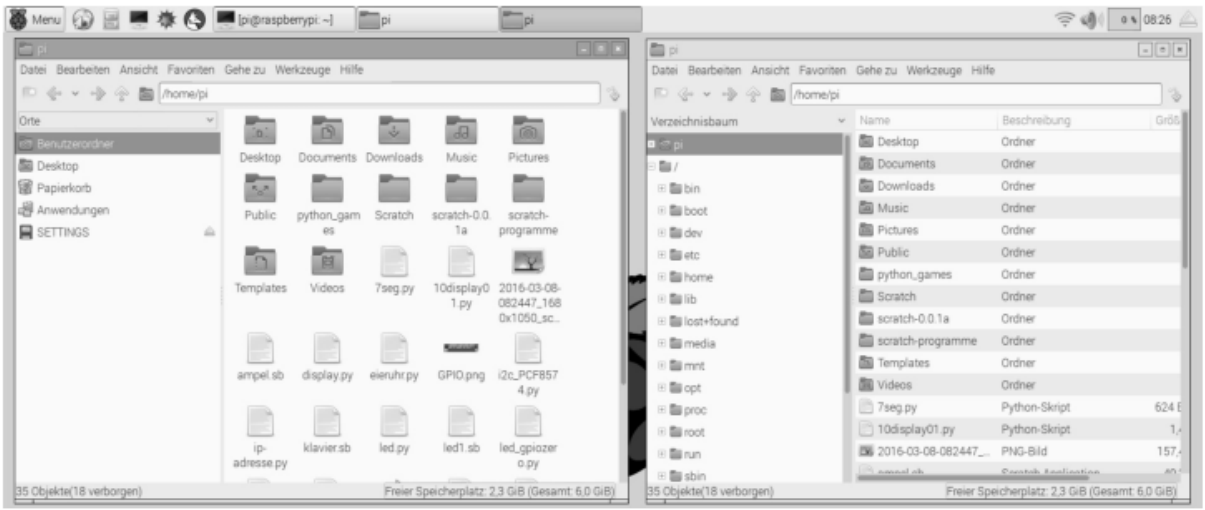
1.3.2 Shranjevanje lastnih datotek na Raspberry Pi

Upravljanje datotek pri operacijskem sistemu Linux sicer poteka malce drugače kot pri operacijskem sistemu Windows, vendar ni nič težje. Raspbian nudi upravitelja datotek, ki je neverjetno podoben programu Windows Explorer. Pomembna razlika v primerjavi z Windows: Linux ne ločuje strogo med pogoni, saj se vse datoteke nahajajo na skupnem datotečnem sistemu.

V operacijskem sistemu Linux vse lastne datoteke načeloma shranjujete samo v lastno domačo mapo (Home). Tukaj se imenuje `/home/pi` v skladu z uporabniškim imenom `pi`. Linux uporablja enostavno poševnico za ločevanje med nivoji map (/) in ne leve poševnice (\), ki jo poznamo pri Windows. V to mapo boste shranjevali tudi svoje programe Python.

Upravitelj datotek v stanju standardnih nastavitev prikazuje tudi samo to domačo mapo. Nekateri programi tam samodejno ustvarijo podmape.

Če želite videti resnično vse, tudi datoteke, ki se običajnega uporabnika ne tičejo, preklopite upravitelja datotek zgoraj levo iz možnosti *Orte (Kraji)* na *Verzeichnisbaum (Drevo map)*. Nato v meniju pod možnostjo *Ansicht (Pogled)* izberite še možnost *Detailansicht (Podroben pogled)* in prikaz je videti tako, kot si to pri operacijskem sistemu Linux tudi predstavljamo.



Slika 1.4: Upravitelj datotek na Raspberry Pi v dveh različnih prikazih.

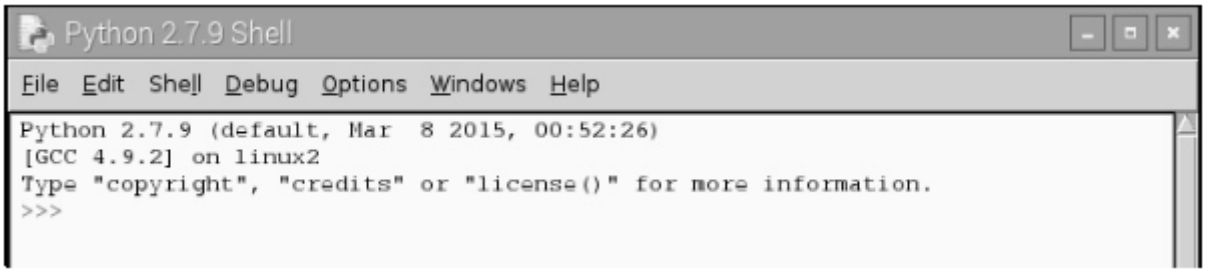
Koliko je nezasedenega prostora na spominski kartici?
Niso samo trdi diski osebnih računalnikov vedno polni – pri spominski kartici računalnika Raspberry Pi lahko do tega pride veliko hitreje. Zato je toliko bolj pomembno, da imate vedno pregled nad nezasedenim in skupnim razpoložljivim prostorom na spominski kartici. Statusna vrstica upravitelja datotek na spodnjem robu okna desno prikazuje nezaseden in zaseden pomnilniški prostor na spominski kartici.

1.4 Prvi program s programskim jezikom Python

Za vstop v svet programiranja je na Raspberry Pi že nameščen programski jezik Python. Python prepriča s svojo jasno strukturo, ki omogoča enostaven začetek programiranja, vendar pa je tudi idealen jezik, ko želite "na hitro" nekaj avtomatizirati, kar bi bilo sicer treba delati ročno. Ker ni treba upoštevati deklaracij spremenljivk, tipov, razredov ali zapletenih pravil, je programiranje resnično zabavno.

Python 2 ali 3?
Na Raspberry Pi sta že nameščeni kar dve različici programskega jezika Python. Žal pa najnovejša različica Python 3.x v nekaterih primerih uporablja drugo sintakso kot uveljavljena različica 2.x, tako da programi iz ene različice ne delujejo v kombinaciji z drugo različico. Nekatere pomembne knjižnice še niso na voljo za Python 3.x. Iz tega razloga in tudi zato, ker je bila v večina programov, ki so na voljo na internetu, napisana za Python 2.x, v tem učnem kompletu uporabljamo uveljavljeno različico Python 2.7.9. Če je na vašem Raspberry Pi nameščena starejša različica Python s številko različice 2.x, potem naši primeri prav tako delujejo v kombinaciji s to različico.

Python 2.7.9 zaženete na namizju prek točke menija *Entwicklung/Python 2 (Razvoj/Python 2)*. Tukaj se pojavi vnosno okno z ukaznim pozivom, ki je na prvi pogled videti preprosto.



Slika 1.5: Vnosno okno pri Python Shell.

V tem oknu odpirate obstoječe programe Python, pišete nove ali pa lahko tudi neposredno interaktivno zaganjate ukaze Python, ne da bi pri tem morali napisati dejanski program. Npr. pri pozivu vnesite naslednje:

```
>>> 1+2
```

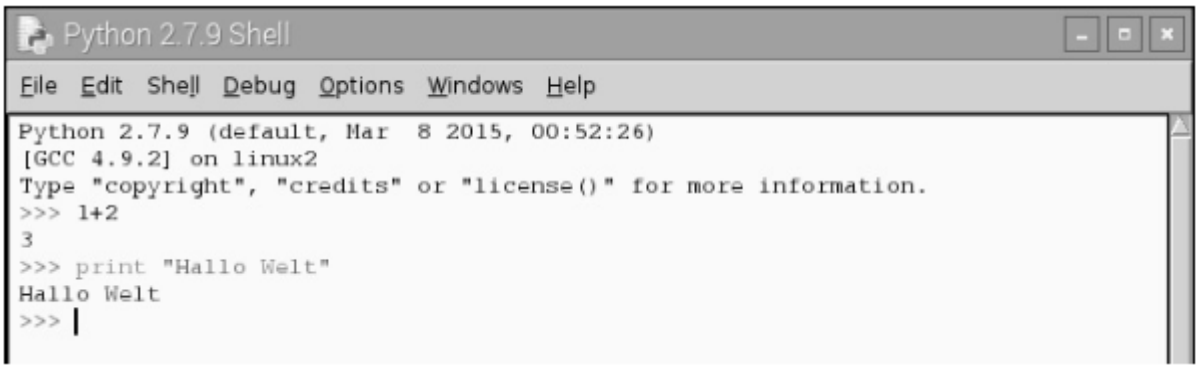
Takoj se pojavi pravilen odgovor:

```
3
```

Na ta način lahko uporabljate Python kot udobni žepni kalkulator, vendar pa pri tem še nimate opravka s programiranjem. Ponavadi se tečaji programiranja začnejo s programom *Pozdravljen svet*, ki na zaslonu izpiše stavek »Pozdravljen svet«. To je v programskem jeziku Python tako enostavno, da se dodeljevanje lastnega naslova sploh ne izplača. V okno Python Shell preprosto vtipkajte naslednjo vrstico:

```
>>> print "Pozdravljen svet"
```

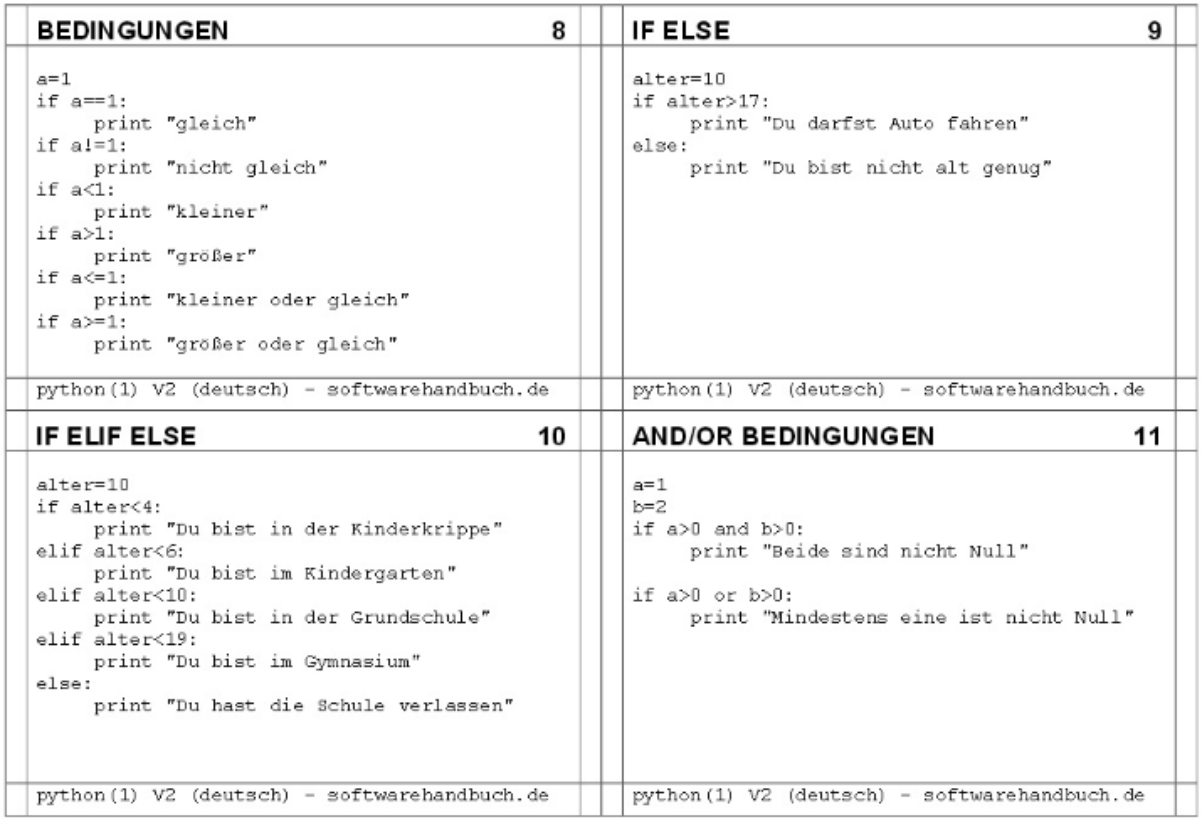
Ta prvi "program" nato zapiše `Pozdravljen svet` v naslednji vrstici na zaslonu.



Slika 1.6: »Pozdravljen svet« v programskem jeziku Python (zgoraj vidite še izpis izračuna).

Tukaj tudi takoj vidite, da Python Shell za ponazoritev samodejno uporablja različne barve besedila. Ukazi Python so oranžne barve, zaporedja znakov so zelene barve, rezultati pa so modre barve. Kasneje boste odkrili še dodatne barve.

Učne kartice za Python
Python je idealni programski jezik za začetnike, ki se želijo naučiti programiranja. Malce se je treba privaditi samo na sintakso in pravila razporeditve. Za pomoč pri vsakdanjem programiranju so najpomembnejši elementi sintakse jezika Python na kratko opisani v obliki majhnih "listkov za plonkanje". Ti temeljijo na učnih karticah za Python, ki jih je pripravil David Whale. Kaj je s tem natančno mišljeno, najdete pri bit.ly/pythonflashcards. Te učne kartice ne razlagajo tehničnih ozadij, temveč samo opisujejo sintakso na podlagi čisto kratkih primerov, torej kako se kaj naredi.



Slika 1.7: Izvleček iz učnih kartic za Python.

1.4.1 Ugibanje števil s Python

Namesto da bi se zadrževali s teorijo o programiranju, algoritmi in tipi podatkov, bomo kar napisali prvo majhno igro v Python. Gre za enostavno igro ugibanja, pri kateri mora igralec v karseda malo korakih uganiti število, ki ga naključno izbere računalnik.

1. V meniju okna Python Shell izberite možnost *File/New Window (Datoteka/Novo okno)*. Pri tem se odpre novo okno, v katerega vtipkate naslednjo programsko kodo:

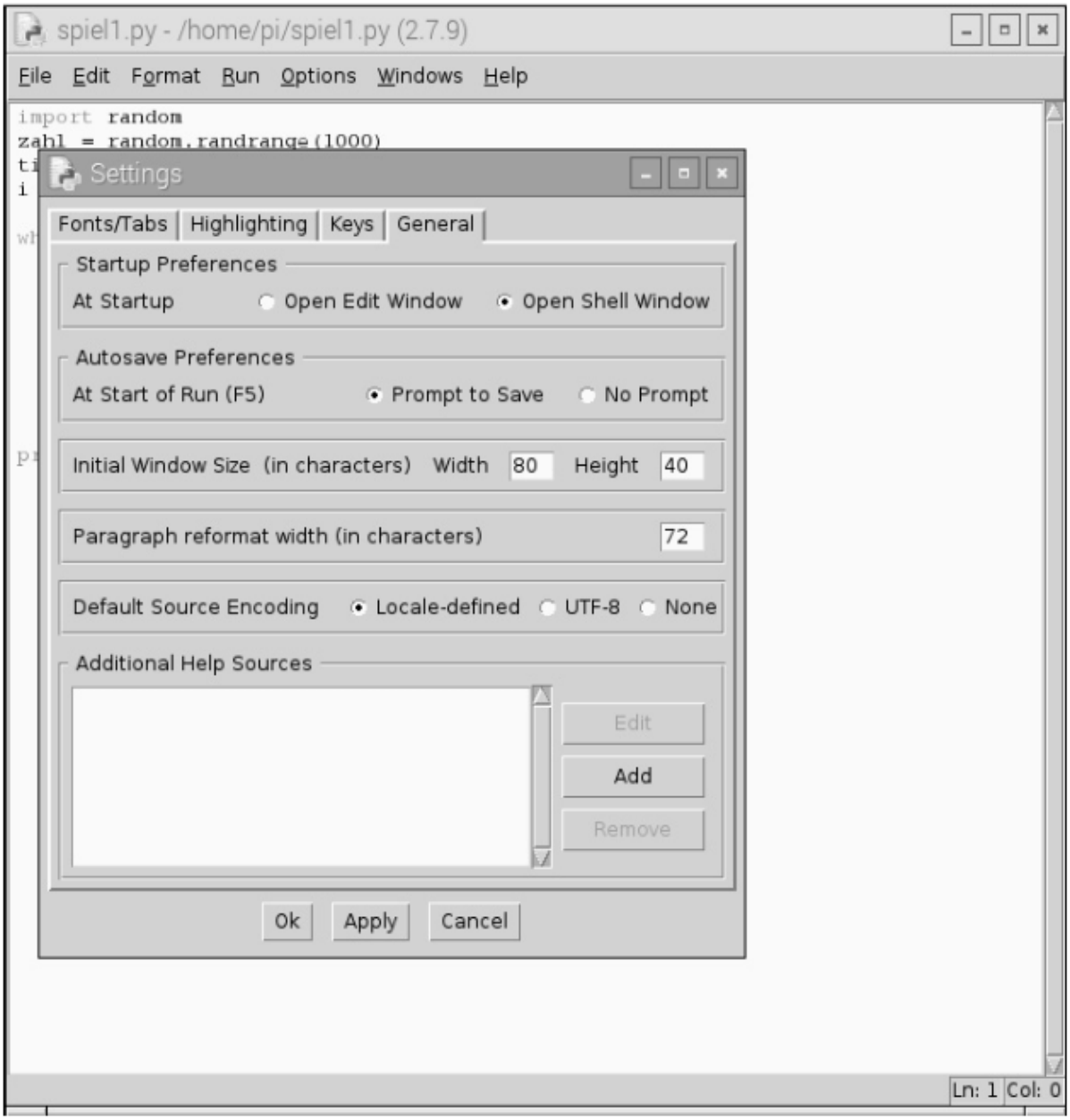
```
import random
zahl = random.randrange(1000); tipp = 0; i = 0
while tipp != zahl:
      tipp = input("Vaše ugibanje:")
      if zahl < tipp:
            print "Iskano število je manjše od ",tipp

      if zahl > tipp:
            print "Iskano število je večje od ",tipp

      i += 1
print "Uganili ste število v ",i,". poskusu"
```

2. Shranite datoteko prek možnosti *File/Save As (Datoteka/Shrani kot)* pod imenom `spiel1.py`. Ali pa si prenesite gotovo programsko datoteko s spletne strani www.buch.cd in jo odprite v Python Shell z možnostjo *File/Open (Datoteka/Odpri)*. Barvno kodiranje v izvorni kodi se samodejno pojavi in pomaga pri iskanju tipkarskih napak.

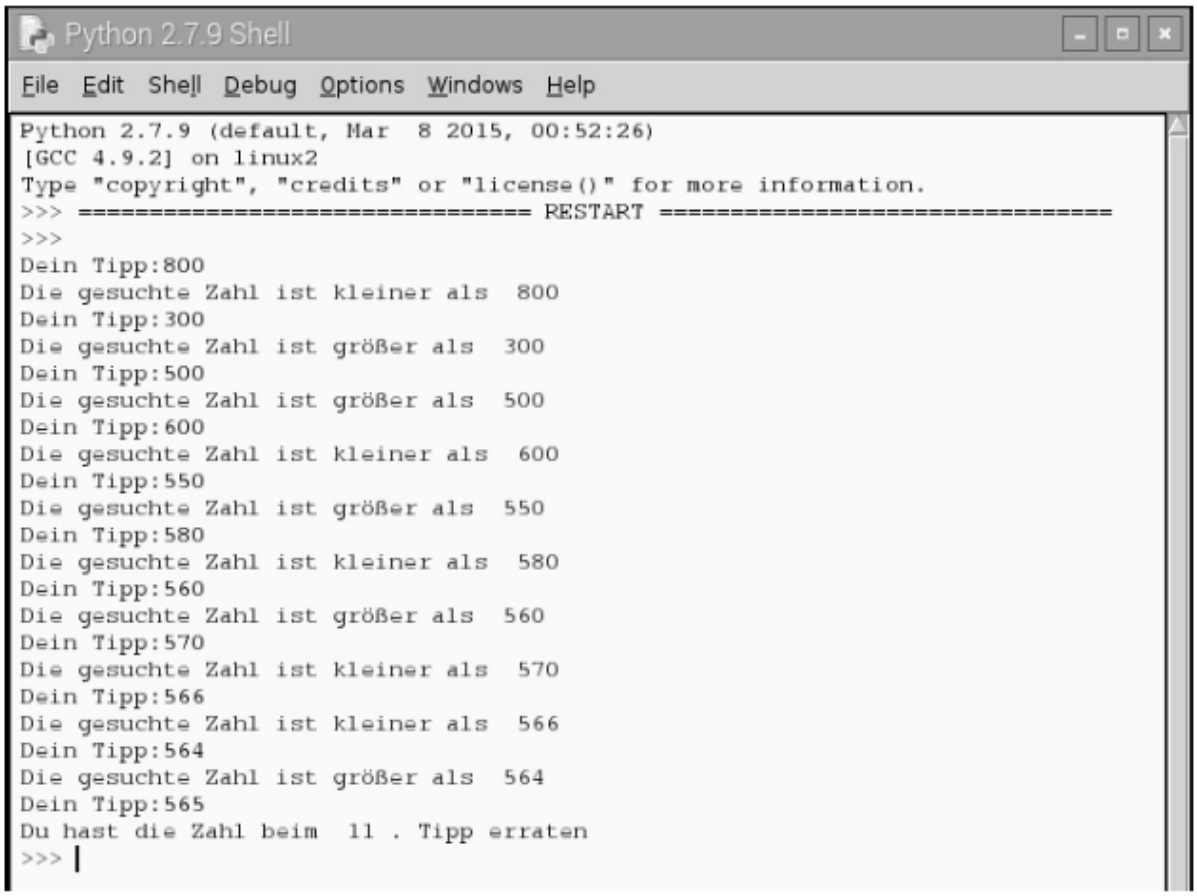
3. Preden aktivirate igro, morate še upoštevati eno posebnost nemškega jezika, in sicer preglase. Python deluje na najrazličnejših računalniških vmesnikih, ki različno kodirajo preglase. Da bodo pravilno prikazani, v meniju izberite *Options/Configure IDLE (Možnosti/Konfiguriraj IDLE)* in v zavihku *General (Splošno)* na območju *Default Source Encoding (Privzeto kodiranje vira)* aktivirajte možnost *Locale-defined (Lokalno določeno)*.



Slika 1.8: Pravilna nastavitev za prikaz preglasov v Python.

4. Sedaj aktivirajte igro s pritiskom tipke [F5] ali izbiro točke menija *Run/Run Module (Zaženi/Zaženi modul)*.

5. V namen enostavnosti igra nima nobenega grafičnega vmesnika, prav tako nima pojasnjevalnih besedil ali verjetnostnih poizvedb glede vnosa. Računalnik v ozadju generira naključno število med 0 in 1.000. Preprosto vnesite poljubno število in izvedeli boste, če je iskano število večje ali manjše. Z nadaljnjimi ugibanji števil se boste postopoma približevali pravilnemu številu.



Slika 1.9: Ugibanje števil v Python.

1.4.2 Tako deluje

Če igra deluje, lahko enostavno preizkusite. Sedaj se seveda pojavi nekaj vprašanj: Kaj se dogaja v ozadju? Kaj pomenijo posamezne programske vrstice?

```
import random
```

Za generiranje naključnega števila se uvozi eksterni Python modul z imenom `random`, ki vsebuje različne funkcije za naključne generatorje.

```
zahl = random.randrange(1000)
```

Funkcija `randrange` iz modula `random` generira naključno število na številskem območju, ki ga omejujejo parametri, tukaj med 0 in 999. Parameter funkcije `random.randrange()` navaja število možnih naključnih števil, ki se začne z 0, torej vedno prvo število, ki ni doseženo. Isto velja tudi za zanke in podobne funkcije v Python.

To naključno število se shrani v spremenljivkah `zahl`. Spremenljivke so v Python spominska mesta, ki imajo poljubno ime in lahko shranjujejo števila, zaporedja znakov, sezname ali druge vrste podatkov. Za razliko od drugih programskih jezikov jih ni treba predhodno deklarirati.

Kako nastajajo naključna števila?
Pogosto velja mnenje, da se v programu ne more nič naključno zgoditi. Kako lahko ima torej program sposobnost generiranja naključnih števil? Če veliko praštevilo delite s kakršnokoli vrednostjo, se po X-tem decimalnem mestu pojavijo števila, ki jih je komajda še možno predvideti. Ta se tudi spreminjajo brez kakršnekoli rednosti, ko delitelj redno povečujemo. Ta rezultat je sicer dozdevno naključen, vendar pa ga je možno z identičnim programom ali z večkratnim priklicem istega programa kadarkoli reproducirati. Če pa sedaj vzamemo število, ki je sestavljeno iz nekaterih od teh številk, in ga spet delimo s številom, ki je rezultat trenutne sekunde prikaza časa ali vsebine poljubnega mesta pomnilnika računalnika, dobimo rezultat, ki ga ni možno reproducirati, zato mu pravimo naključno število.

`tipp = 0`
Spremenljivka `tipp` kasneje vsebuje število, ki ga vtipka uporabnik. Na začetku je `0`.

`i = 0`
Spremenljivka `i` se je med programerji uveljavila kot števec za prehode programske zanke. Tukaj se uporablja za štetje poskusov ugibanja, ki jih je uporabnik potreboval za ugotavljanje skritega števila. Tudi ta spremenljivka se na začetku nahaja na `0`.

`while tipp != zahl:`
Beseda `while` (angleško za »dokler«) uvede programsko zanko, ki se v tem primeru tako dolgo ponavlja, dokler je `tipp` (število, ki ga vtipka uporabnik) neenak skritemu številu `zahl`. Python uporablja kombinacijo `!=` za neenako. Za dvopičjem sledi dejanska programska zanka.

`tipp = input("Vaše ugibanje:")`
Funkcija `input` napiše besedilo `Vaše ugibanje:`, nato pa pričakuje vnos, ki se shrani v spremenljivki `tipp`.

Zamiki so pomembni v Python
Pri večini programskih jezikov so programske zanke ali odločitve zamaknjene, tako da je programska koda preglednejša. V Python ti zamiki ne služijo samo preglednosti, temveč so tudi obvezno potrebni za programsko logiko. Zato pri tem niso potrebna posebna ločila za zaključitev zank ali odločitev.

`if zahl < tipp:`
Če je skrito število `zahl` manjše od števila `tipp`, ki ga je vtipkal uporabnik, potem ...

```
print "Iskano število je manjše od ",tipp
```

... je prikazano to besedilo. Na koncu se tukaj nahaja spremenljivka `tipp`, da je vtipkano število prikazano v besedilu. Če ta pogoj ne velja, se zamaknjena vrstica enostavno preskoči.

`if tipp < zahl:`
Če je skrito število `zahl` večje od števila `tipp`, ki ga je vtipkal uporabnik, potem ...

```
print "Iskano število je večje od ",tipp
```

... je prikazano drugo besedilo.

`i += 1`
V vsakem primeru – zato tudi ni več zamaknjen – se števec `i`, ki šteje poskuse ugibanja, poveča za 1. Ta vrstica z operatorjem `+=` pomeni isto kot `i = i + 1`.

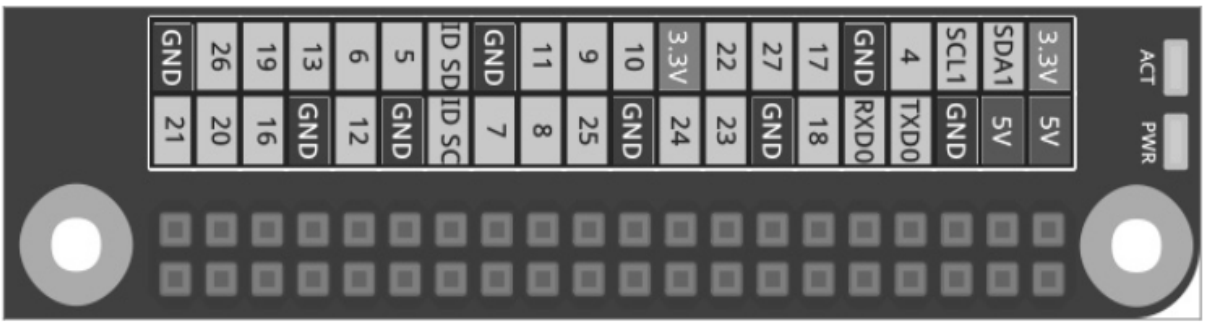
```
print "Uganili ste število v ",i,". poskusu"
```

Ta vrstica je bolj zamaknjena, kar pomeni, da je tudi zanka `while` končana. Ko torej njen pogoj več ne drži, torej število `tipp`, ki ga je vtipkal uporabnik, več ni neenako (temveč je enako) skritemu številu `zahl`, se pojavi to besedilo, ki je sestavljeno iz dveh stavčnih delov in spremenljivke `i` ter tako navaja, koliko poskusov je uporabnik potreboval. Programi Python ne potrebujejo lastnega navodila za ustavitev. Ustavijo se preprosto po zadnjem ukazu oz. po zanki, ki se več ne izvaja in kateri več ne sledijo nadaljnji ukazi.

2 Na Raspberry Pi sveti prva LED

40-pinska letev z moškimi kontakti na robu Raspberry Pi nudi možnost neposredne priključitve strojne opreme, tako da lahko npr. prek tipkal izvajate vnose ali pa aktivirate LED s krmiljenjem prek programa. Tej letvi z moškimi kontakti pravimo GPIO. Angleška kratica GPIO (General Purpose Input Output) pomeni »Splošni vhod in izhod«.

Od teh 40 pinov lahko 22, ki so označeni samo s številkami, po želji programirate kot vhod ali izhod in jih tako uporabljate za raznolike nadgradnje strojne opreme. Ostali so fiksno nastavljeni za napajanje in druge namene.



Slika 2.1: Razporeditev GPIO-vmesnika. Zaokrožen rob spodaj desno se sklada z robom vezja Raspberry Pi.

Previdno

V nobenem primeru ne smete medsebojno povezovati katerihkoli GPIO-pinov in čakati, kaj se bo zgodilo, temveč obvezno upoštevajte naslednje napotke:

Nekateri GPIO-pini so neposredno povezani s priključki procesorja, kratek stik lahko popolnoma uniči Raspberry Pi. Če prek stikala ali LED medsebojno povežete dva pina, potem morate med njiju vedno priključiti predupor.

Za logične signale vedno uporabljajte pin 1, ki nudi +3,3 V napetost in ga lahko obremenite s tokom do 50 mA. Vsi pini, ki so označeni z napisom GND, so vodniki za maso za logične signale.

Vsak GPIO-pin lahko preklopite kot izhod (npr. za LED) ali kot vhod (npr. za tipkala). GPIO-izhodi v logičnem stanju 1 nudijo napetost +3,3 V, v logičnem stanju 0 pa 0 V. GPIO-vhodi pri napetosti do +1,7 V nudijo logični signal 0, pri napetosti med +1,7 V in +3,3 V pa logični signal 1.

Pin 2 in pin 4 nudita +5 V napetost za napajanje zunanje strojne opreme. Tukaj je možno odjemati toliko toka, kolikor ga nudi USB-napajalnik računalnika Raspberry Pi. Vendar pa teh pinov ne smete povezati z GPIO-vhodom.

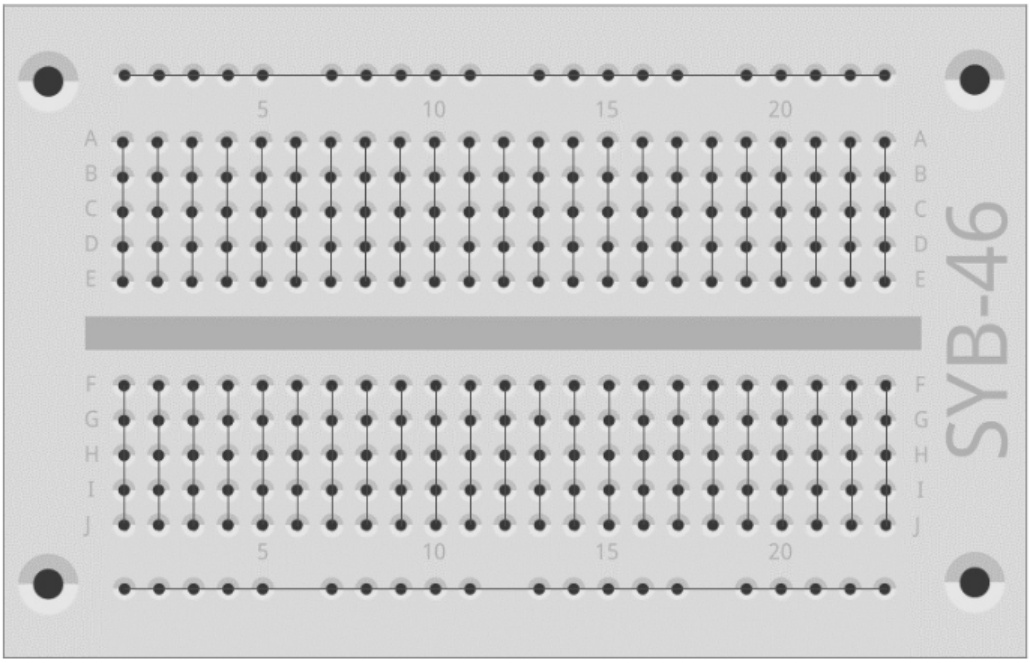
2.1 Komponente v paketu

Učni komplet vsebuje različne elektronske komponente, s katerimi lahko sestavite preizkuse, ki so opisani v teh navodilih za uporabo (in seveda tudi lastne preizkuse). Na tem mestu so komponente samo na kratko predstavljene. Potrebne praktične izkušnje pri rokovanju z njimi si boste nato pridobili z dejanskimi preizkusi.

- 2 x preizkusna ploščica
- 1 x rdeča LED
- 1 x rumena LED
- 1 x zelena LED
- 1 x modra LED
- 4 x tipkalo
- 4 x upor 10 kΩ (rjav - črn - oranžen)
- 4 x upor 1 kΩ (rjav - črn - rdeč)
- 4 x upor 220 kΩ (rdeč - rdeč - rjav)
- 12 x priključni kabel (preizkusna ploščica – Raspberry Pi)
- Pribl. 1 m stikalna žica

2.1.1 Preizkusni ploščici

Za hitro sestavljanje elektronskih vezij sta kompletu priloženi dve preizkusni ploščici. Pri tem lahko elektronske komponente neposredno vstavite v raster z luknjicami s standardnimi razmaki, tako da vam ni treba spajkati. Pri teh preizkusnih ploščicah so zunanje vzdolžne vrste s kontakti (X in Y) vse medsebojno povezane.



Slika 2.2: Preizkusna ploščica iz kompleta z vgrajenimi povezavami.

Te kontaktne vrste se pogosto uporabljajo kot plus in minus pol za napajanje vezij. V drugih kontaktnih vrstah je po pet kontaktov (A do E in F do J) prečno povezanih med seboj, pri

čemer je na sredini preizkusne ploščice prazen prostor. Tako lahko tukaj na sredini vstavite večje komponente in jih ožičite navzven.

2.1.2 Priključni kabli

Barvni priključni kabli imajo vsi na eni strani majhen žični vtič, s katerim jih lahko vstavite v preizkusno ploščico. Na drugi strani se nahaja vtični priključek, ki je primeren za priključitev na GPIO-pin računalnika Raspberry Pi.

Poleg tega je učnemu kompletu priložena stikalna žica. Iz nje lahko izdelate kratke mostičke, s katerimi so nato povezane kontaktne vrste na preizkusni ploščici. Odrežite žico z majhnimi kleščami ščipalkami na ustrezne dolžine v skladu z opisom pri posameznih preizkusih. Da lahko žice bolje vstavite v preizkusno ploščico, je priporočljivo, da jih rahlo poševno odrežete, tako da dobite obliko klina. Na obeh koncih odstranite izolacijo dolžine približno pol centimetra.

2.1.3 Upori in njihove barvne kode

Upori se v digitalni elektroniki v glavnem uporabljajo za omejitev toka na vratih mikrokrmilnika ter kot predupori za LED. Merska enota za upore je Ω (ohm). 1.000 Ω je 1 kΩ (kiloohm).

Vrednosti upornosti so označene na uporih z barvnimi obroči. Večina uporov ima štiri takšne barvne obroče. Prva dva barvna obroča se nanašata na številke, tretji označuje množitelj, četrti pa toleranco. Ta obroč za toleranco je ponavadi zlate ali srebrne barve. Pri tem gre za barvi, ki se na prvih obročih ne pojavljajo, tako da je smer branja jasna. Sama vrednost tolerance pri digitalni elektroniki skorajda ne igra nobene vloge.

Barva	**Vrednost upornosti v Ω**			
	1. obroč (desetica)	**2. obroč (enica)**	**3. obroč (množitelj)**	**4. obroč (toleranca)**
Srebrna			$10^{-2} = 0{,}01$	±10 %
Zlata			$10^{-1} = 0{,}1$	±5 %
Črna		0	$10^{0} = 1$	
Rjava	1	1	$10^{1} = 10$	±1 %
Rdeča	2	2	$10^{2} = 100$	±2 %
Oranžna	3	3	$10^{3} = 1.000$	
Rumena	4	4	$10^{4} = 10.000$	
Zelena	5	5	$10^{5} = 100.000$	±0,5 %
Modra	6	6	$10^{6} = 1.000.000$	±0,25 %
Vijolična	7	7	$10^{7} = 10.000.000$	±0,1 %
Siva	8	8	$10^{8} = 100.000.000$	±0,05 %
Bela	9	9	$10^{9} = 1.000.000.000$	

Tabela 2.1: Tabela prikazuje pomen barvnih obročev na uporih.

Učni komplet vsebuje upore v treh različnih vrednostih:

Vrednost	1. obroč (desetica)	2. obroč (enica)	3. obroč (množitelj)	4. obroč (toleranca)	Uporaba
220 Ω	Rdeča	Rdeča	Rjava	Zlata	Predupor za LED
1 kΩ	Rjava	Črna	Rdeča	Zlata	Zaščitni upor za GPIO-vhode
10 kΩ	Rjava	Črna	Oranžna	Zlata	Pull-down upor oz. spodnji upor za GPIO-vhode

Tabela 2.2: Barvne kode uporov v učnem kompletu.

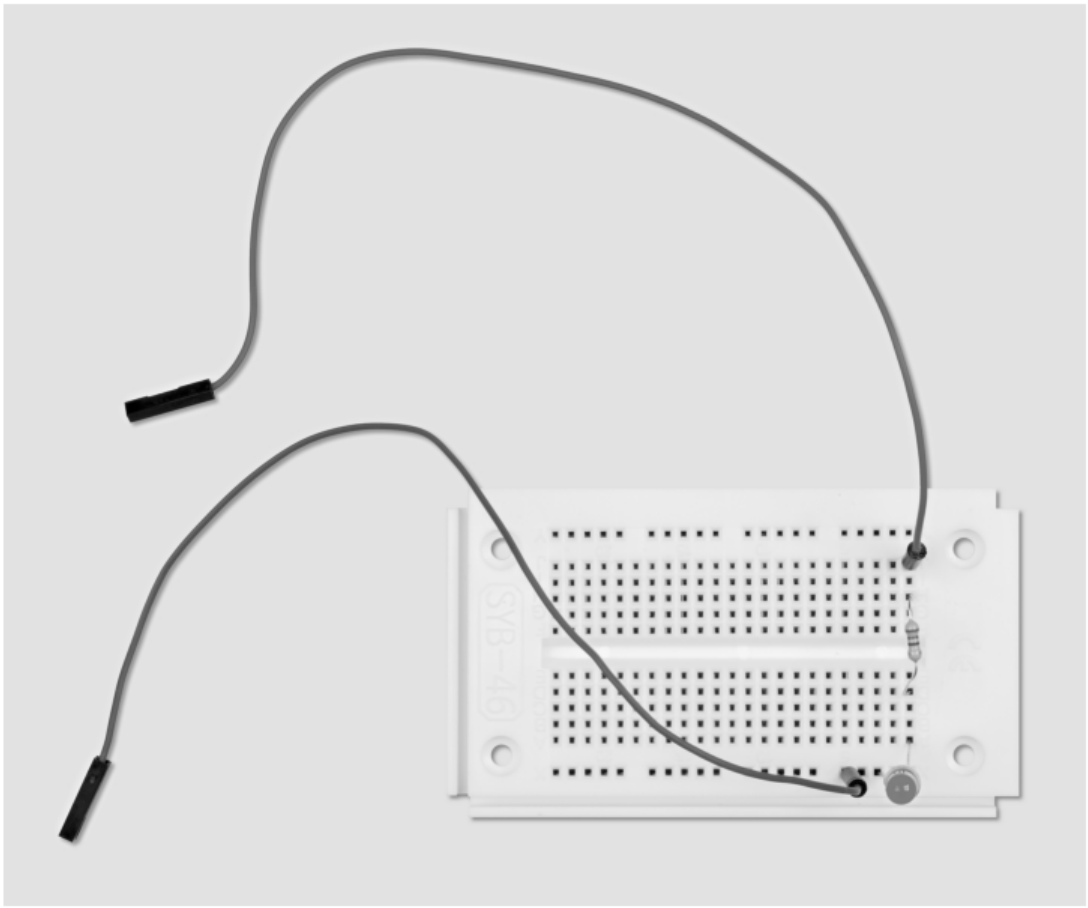
Predvsem pri 1 kΩ in 10 kΩ uporih bodite zelo pozorni na barve, saj jih lahko hitro zamenjate.

2.2 Priključitev LED

Na GPIO-vrata lahko priključite LED za svetlobne signale in svetlobne učinke (LED = Light Emitting Diode = svetleča dioda). Pri tem je treba med uporabljen GPIO-pin in anodo LED vgraditi 220 Ω predupor (rdeč-rdeč-rjav), ki služi omejitvi prevodnega toka in s tem preprečitvi pregoretja LED. Predupor dodatno ščiti tudi GPIO-izhod računalnika Raspberry Pi, saj LED v prevodni smeri ne nudi skoraj nobene upornosti, zato se lahko GPIO-vrata pri povezavi z maso hitro preobremenijo. Katodo LED je treba povezati z vodnikom za maso na pin 6.

V kateri smeri je treba priključiti LED?
Priključni žici pri LED sta različno dolgi. Daljša izmed njiju je plus pol oz. anoda, krajša pa je katoda. To si je enostavno zapomniti: Znak plus ima eno črtico več kot znak minus, s čimer je žica malce daljša. Poleg tega je večina LED na minus strani sploščenih, tako kot znak minus. Enostavno si je zapomniti: katoda = kratka = rob.

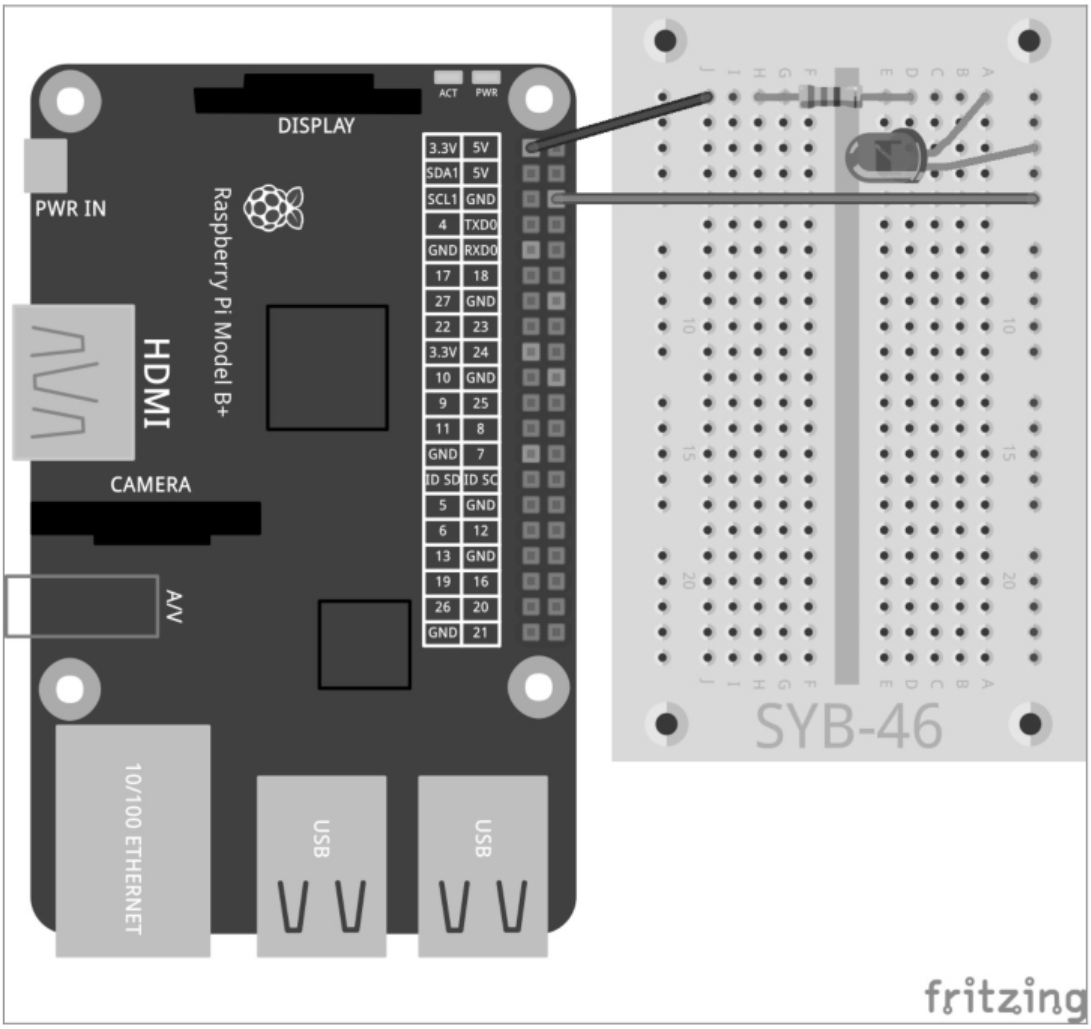
Najprej tako kot prikazuje slika priključite LED prek 220 Ω predupora (rdeč-rdeč-rjav) na +3,3 V priključek (pin 1) in povežite minus pol LED z vodnikom za maso (pin 6).



Slika 2.3: Sestav preizkusne ploščice za priključitev LED.

Potrebne komponente:

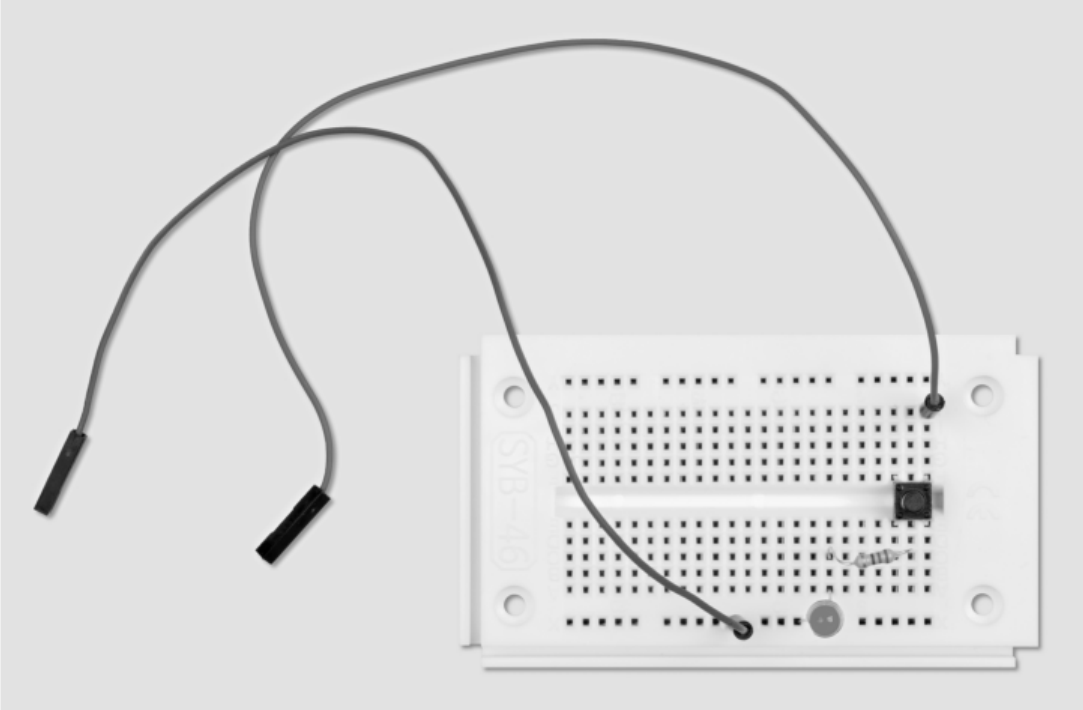
1 x preizkusna ploščica
1 x rdeča LED
1 x 220 Ω upor
2 x priključni kabel



Slika 2.4: Prva LED na Raspberry Pi.

V tem prvem preizkusu se Raspberry Pi uporablja samo kot napajanje za LED. LED vedno sveti in za to ne potrebujete nobene programske opreme.

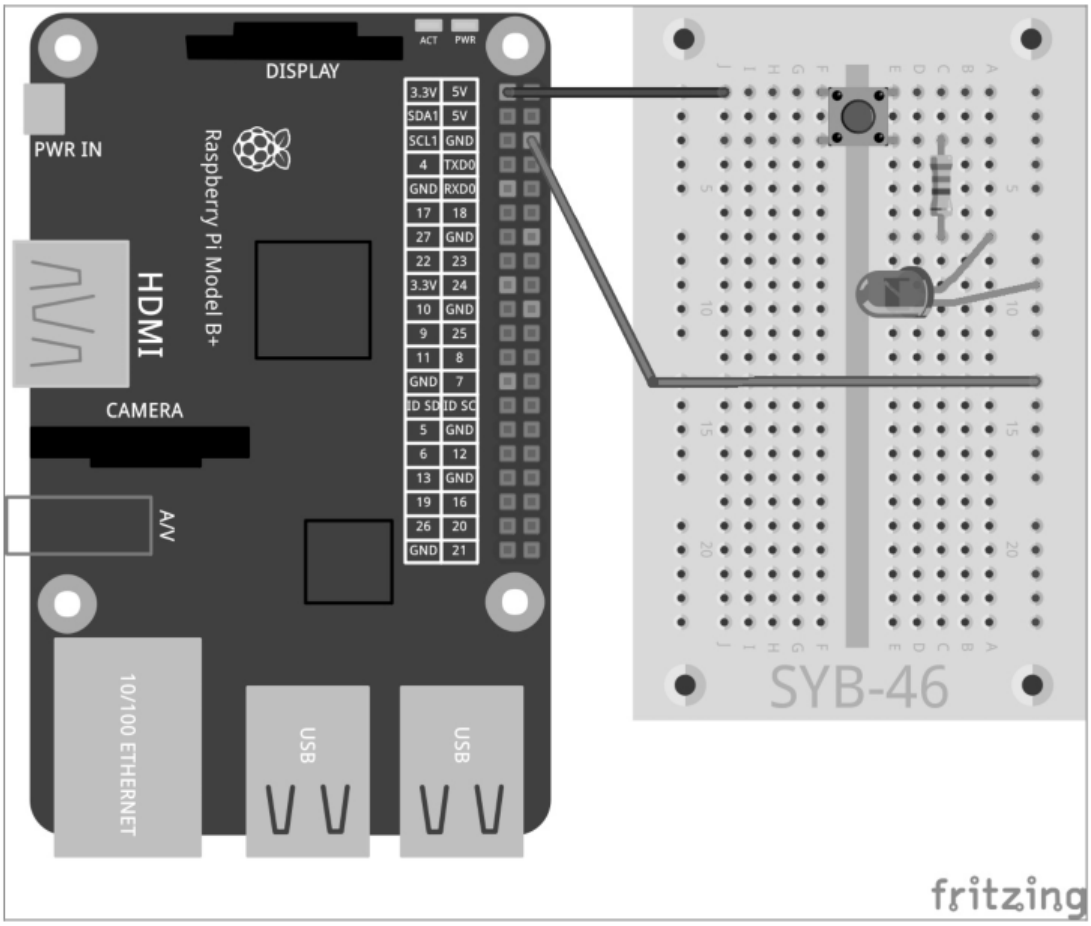
V naslednjem preizkusu v napajalni kabel LED vstavite tipkalo. LED sedaj sveti samo takrat, ko je to tipkalo pritisnjeno. Tudi za to ne potrebujete programske opreme.



Slika 2.5: Sestav preizkusne ploščice za LED, ki jo preklapljate s stikalom.

Potrebne komponente:

1 x preizkusna ploščica
1 x rdeča LED
1 x 220 Ω upor
1 x tipkalo
2 x priključni kabel



Slika 2.6: LED s tipkalom na Raspberry Pi.

2.3 GPIO s Python

Da lahko uporabljate GPIO-vrata prek programov Python, mora biti nameščena knjižnica Python GPIO. V zgodnejših različicah Raspbian so morali uporabniki to knjižnico še ročno nameščati. To danes več ni potrebno, pa tudi dostop "sudo", ki je bil včasih potreben, je sedaj že stvar preteklosti.

2.4 Vklapljanje in izklapljanje LED

Tako kot prikazuje naslednja slika priključite LED prek 220 Ω predupora (rdeč-rdeč-rjav) na GPIO-vrata 25 (pin 22) in ne več neposredno na +3,3 V priključek ter povežite minus pol LED prek letve za izenačitev potencialov preizkusne ploščice z vodnikom za maso Raspberry Pi (pin 6).

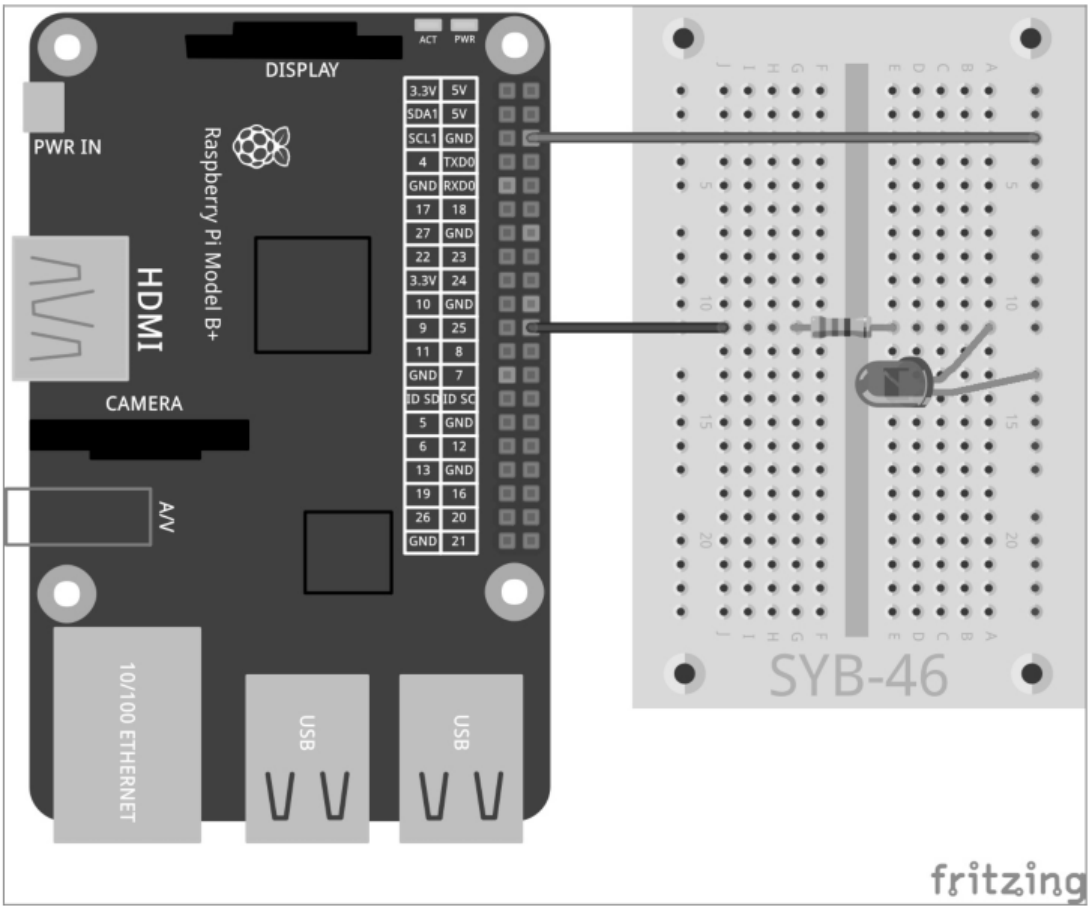
Potrebne komponente:

1 x preizkusna ploščica
1 x rdeča LED
1 x 220 Ω upor
2 x priključni kabel

Naslednji program `led.py` za 2 sekundi vklopi LED in jo nato spet izklopi:

```
import RPi.GPIO as GPIO
import time
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(25, GPIO.OUT)
GPIO.output(25, 1)
time.sleep(2)
GPIO.output(25, 0)
GPIO.cleanup()
```



Slika 2.7: LED na GPIO-vratih 25.

2.4.1 Tako deluje

Primer prikazuje najpomembnejše osnovne funkcije knjižnice `RPi.GPIO`.

`import RPi.GPIO as GPIO`
Knjižnico `RPi.GPIO` je treba uvoziti v vsak program Python, v katerem jo nameravate uporabljati. S tem načinom pisanja je možno krmiljenje vseh funkcij knjižnice prek predpone `GPIO`.

`import time`
Pogosto uporabljena knjižnica Python `time` nima nobene veze z GPIO-programiranjem. Vsebuje funkcije za izračun časa in datuma, med drugim tudi funkcijo `time.sleep`, s katero je možno na enostaven način udejanjiti čakalne čase v programu.

`GPIO.setmode(GPIO.BCM)`
Na začetku vsakega programa je treba določiti, kako so GPIO-vrata označena. Ponavadi se uporablja standardno oštevilčenje `BCM`.

Oštevilčenje GPIO-vrat
Knjižnica `RPi.GPIO` podpira dve različni metodi za označevanje vrat. V načinu `BCM` se uporabljajo znane številke GPIO-vrat, ki se uporabljajo tudi na nivoju ukaznih vrstic ali v skriptih Shell. V alternativnem načinu `BOARD` se oznake skladajo s številkami pinov od 1 do 40 na vezju Raspberry Pi.

```
GPIO.setup(25, GPIO.OUT)
```
Funkcija `GPIO.setup` inicializira GPIO-vrata kot izhod ali kot vhod. Prvi parameter označi vrata v odvisnosti od izbranega načina `BCM` ali `BOARD` z njihovo GPIO-številko ali s pin številko. Drugi parameter je lahko `GPIO.OUT` za izhod ali `GPIO.IN` za vhod.

```
GPIO.output(25, 1)
```
Na pravkar inicializiranih vratih se pojavi `1`. LED, ki je priključena nanje, sveti. Namesto `1` sta na izhodu možni tudi predhodno določeni vrednosti `True` ali `GPIO.HIGH`.

```
time.sleep(2)
```
Ta funkcija iz knjižnice `time`, ki ste jo uvozili na začetku programa, določa čakalni čas 2 sekundi, preden program teče naprej.

```
GPIO.output(25, 0)
```
Za izklop LED je treba na GPIO-vratih vnesti vrednost `0` oz. `False` ali `GPIO.LOW`.

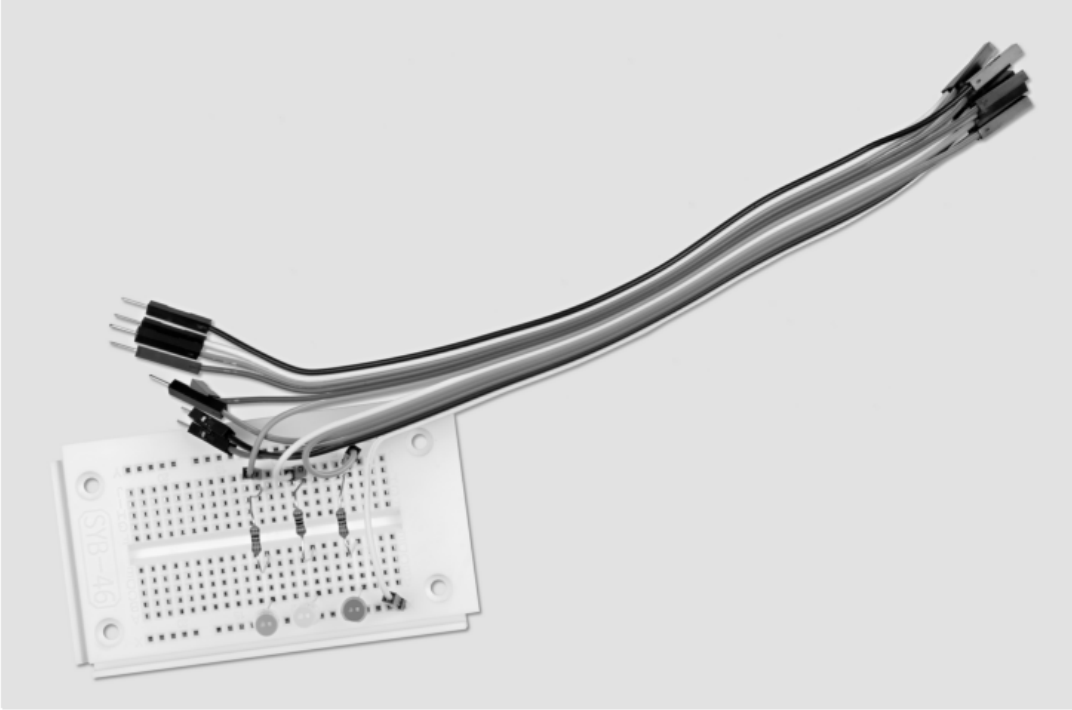
```
GPIO.cleanup()
```
Na koncu programa je treba vsa GPIO-vrata spet ponastaviti. Ta vrstica to naenkrat opravi za vse GPIO-vrata, ki jih je inicializiral program. Vrata, ki so jih inicializirali drugi programi, ostanejo nespremenjena. Tako potek teh drugih programov, ki morda vzporedno potekajo, ni moten.

Prestrezanje GPIO-opozoril
Če želite konfigurirati GPIO-vrata, ki niso bila čisto ponastavljena, temveč so še morda odprta v drugem programu ali v prekinjenem programu, prihaja do opozoril, ki pa ne prekinejo toka programa. Med razvijanjem programa so lahko ta opozorila zelo koristna za odkrivanje napak. V gotovem programu pa lahko neizkušenega uporabnika zmedejo. Iz tega razloga GPIO-knjižnica z `GPIO.setwarnings(False)` nudi možnost izločevanja teh opozoril.

3 Semafor

Vklapljanje in ponovno izklapljanje ene same LED je lahko na prvi pogled zelo zanimivo, vendar pa za to dejansko ne potrebujemo računalnika. Semafor z njegovim značilnim svetlobnim ciklom od zelene, čez rumeno do rdeče in nato s kombinacijo luči rdeča-rumena spet nazaj do zelene je možno s tremi LED enostavno sestaviti in demonstrira nadaljnje tehnike programiranja v Python.

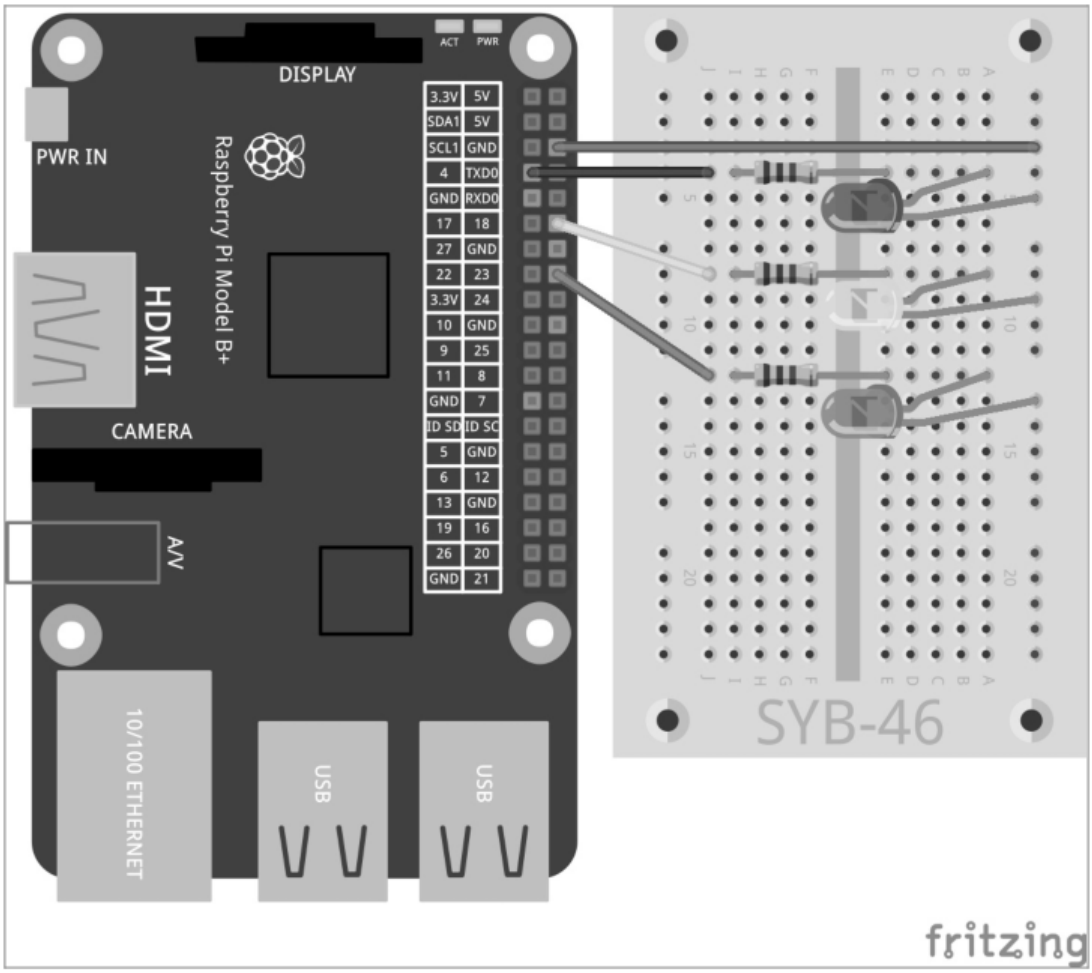
Na preizkusni ploščici sestavite vezje, ki ga vidite na sliki. Za krmiljenje LED se uporabljajo tri GPIO-vrata in skupni vodnik za maso. Številke GPIO-vrat v načinu BCM so navedene na risbi na Raspberry Pi.



Slika 3.1: Sestav preizkusne ploščice za semafor.

Potrebne komponente:

1 x preizkusna ploščica
1 x rdeča LED
1 x rumena LED
1 x zelena LED
3 x 220 Ω upor
4 x priključni kabel



Slika 3.2: Enostaven semafor.

Program `ampel01.py` krmili semafor:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
rot = 0; gelb = 1; gruen = 2
Ampel=[4,18,23]
GPIO.setup(Ampel[rot], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gelb], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gruen], GPIO.OUT, initial=True)
print ("Ctrl+C zaključi program")
try:
      while True:
            time.sleep(2)
            GPIO.output(Ampel[gruen],False);
            GPIO.output(Ampel[gelb],True)
            time.sleep(0.6)
            GPIO.output(Ampel[gelb],False);
            GPIO.output(Ampel[rot],True)
            time.sleep(2)
            GPIO.output(Ampel[gelb],True)
            time.sleep(0.6)
            GPIO.output(Ampel[rot],False);
            GPIO.output(Ampel[gelb],False)
```

```
            GPIO.output(Ampel[gruen],True)
except KeyboardInterrupt:
      GPIO.cleanup()
```

3.1.1 Tako deluje

Prve vrstice so že znane in uvozijo knjižnice `RPi.GPIO` za krmiljenje GPIO-vrat in `time` za časovne zakasnitve. Nato sledi oštevilčenje GPIO-vrat na BCM v skladu s prejšnjim primerom.

`rot = 0; gelb = 1; gruen = 2`
Te vrstice definirajo tri spremenljivke `rot`, `gelb` in `gruen` za tri LED (rdeča, rumena in zelena). S tem si vam v programu ni treba zapomniti številk ali GPIO-vrat, temveč lahko LED enostavno krmilite glede na njihove barve.

`Ampel=[4,18,23]`
Za krmiljenje treh LED ustvarite seznam, ki vsebuje GPIO-številke v zaporedju, v katerem so LED vgrajene na preizkusni ploščici. Ker se GPIO-vrata pojavijo v programu samo na tem mestu, lahko program popolnoma enostavno predelate, če želite uporabiti druga GPIO-vrata.

```
GPIO.setup(Ampel[rot], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gelb], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gruen], GPIO.OUT, initial=True)
```

Zaporedoma se tri uporabljena GPIO-vrata inicializirajo kot izhodi. Pri tem tokrat ne uporabite številk GPIO-vrat, temveč predhodno definiran seznam. Znotraj seznama so posamezni elementi indicirani s številkami, ki se začnejo z 0. `Ampel[0]` je torej prvi element, v tem primeru `4`. Spremenljivke `rot`, `gelb` in `gruen` vsebujejo številke `0`, `1` in `2`, ki so potrebne kot indici za elemente seznama. Na ta način je možno uporabljena GPIO-vrata naslavljati prek barv:

- `Ampel[rot]` se sklada z GPIO-vrati `4` z rdečo LED.
- `Ampel[gelb]` se sklada z GPIO-vrati `18` z rumeno LED.
- `Ampel[gruen]` se sklada z GPIO-vrati `23` z zeleno LED.

Navodila `GPIO.setup` lahko vsebujejo dodatni parameter `initial`, ki GPIO-vratom že pri inicializaciji dodeli logični status. S tem v tem programu že od začetka vklopite zeleno LED. Drugi dve LED začneta program v izklopljenem stanju.

`print ("Ctrl+C zaključi program")`
Sedaj se na zaslonu pojavijo kratka navodila za uporabo. Program deluje samodejno. Kombinacija tipk `[Ctrl]+[C]` je namenjena njegovi zaključitvi. Za poizvedbo, če lahko uporabnik s `[Ctrl]+[C]` zaključi program, uporabite poizvedbo `try...except`. Pri tem se programska koda, ki je vnesena pod `try:`, najprej izvede na običajen način. Če med tem pride do sistemske izjeme – to je lahko napaka ali tudi kombinacija tipk `[Ctrl]+[C]`, pride do prekinitve in izvede se navodilo `except` na koncu programa.

```
except KeyboardInterrupt:
      GPIO.cleanup()
```

S to kombinacijo tipk se sproži `KeyboardInterrupt`, zanka pa se samodejno zapusti. Zadnja vrstica zapre uporabljena GPIO-vrata in s tem izklopi vse LED. Nato se program zaključi. Z nadzorovanim zapiranjem GPIO-vrat ne prihaja do sistemskih opozoril ali sporočil o prekinitvi, ki bi lahko zmedla uporabnika. Dejanski cikel semaforja poteka v neskončni zanki:

`while True:`
Takšne neskončne zanke vedno potrebujejo pogoj za prekinitev, sicer se program nikoli ne bi zaključil.

```
time.sleep(2)
```
Na začetku programa in tudi pri vsakem novem začetku zanke 2 sekundi sveti zelena LED.

```
GPIO.output(Ampel[gruen],False); GPIO.output(Ampel[gelb],True)
time.sleep(0.6)
```

Sedaj se zelena LED izklopi, namesto nje pa se vklopi rumena LED. Ta nato 0,6 sekunde sama sveti.

```
GPIO.output(Ampel[gelb],False); GPIO.output(Ampel[rot],True)
time.sleep(2)
```

Sedaj se rumena LED spet izklopi, namesto nje pa se vklopi rdeča LED. Ta nato 2 sekundi sama sveti. Rdeča faza semaforja je ponavadi občutno daljša od rumene faze.

```
GPIO.output(Ampel[gelb],True)
time.sleep(0.6)
```

Na začetku rdeče-rumene faze se rumena LED dodatno vklopi, pri tem pa se druga LED ne izklopi. Ta faza traja 0,6 sekunde.

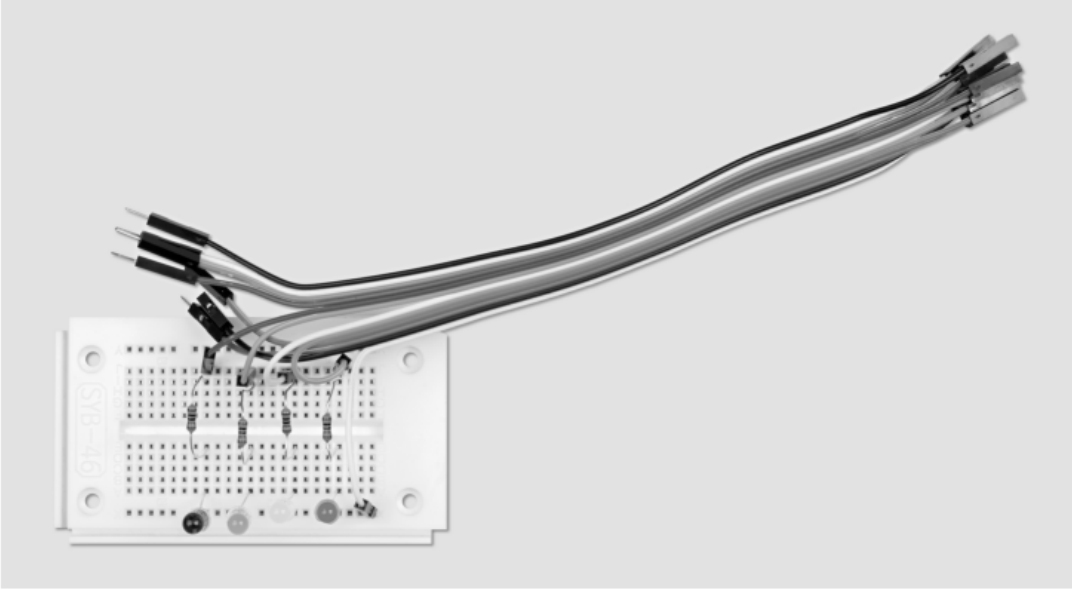
```
GPIO.output(Ampel[rot],False)
GPIO.output(Ampel[gelb],False)
GPIO.output(Ampel[gruen],True)
```

Na koncu zanke semafor spet skoči na zeleno. Rdeča in rumena LED se izklopita, zelena pa se vklopi. V zeleni fazi semaforja se zanka začne od začetka s čakalnim časom 2 sekund. Seveda lahko vse čase poljubno prilagodite. V resničnem svetu so faze semaforja odvisne od dimenzij križišča in od prometnih tokov. Rumena in rdeče-rumena faza sta ponavadi dolgi 2 sekundi.

4 Semafor za pešce

V naslednjem preizkusu bomo vezje s semaforjem razširili z dodatnim semaforjem za pešce, ki med rdečo fazo semaforja prikazuje utripajočo luč za pešce, tako kot se ta uporablja v nekaterih državah. Seveda bi lahko v program vgradili tudi semafor za pešce z rdečo in zeleno lučjo, ki je značilen za Srednjo Evropo, vendar pa ta učni komplet poleg LED, ki jih uporablja semafor, vključuje samo še eno dodatno LED.

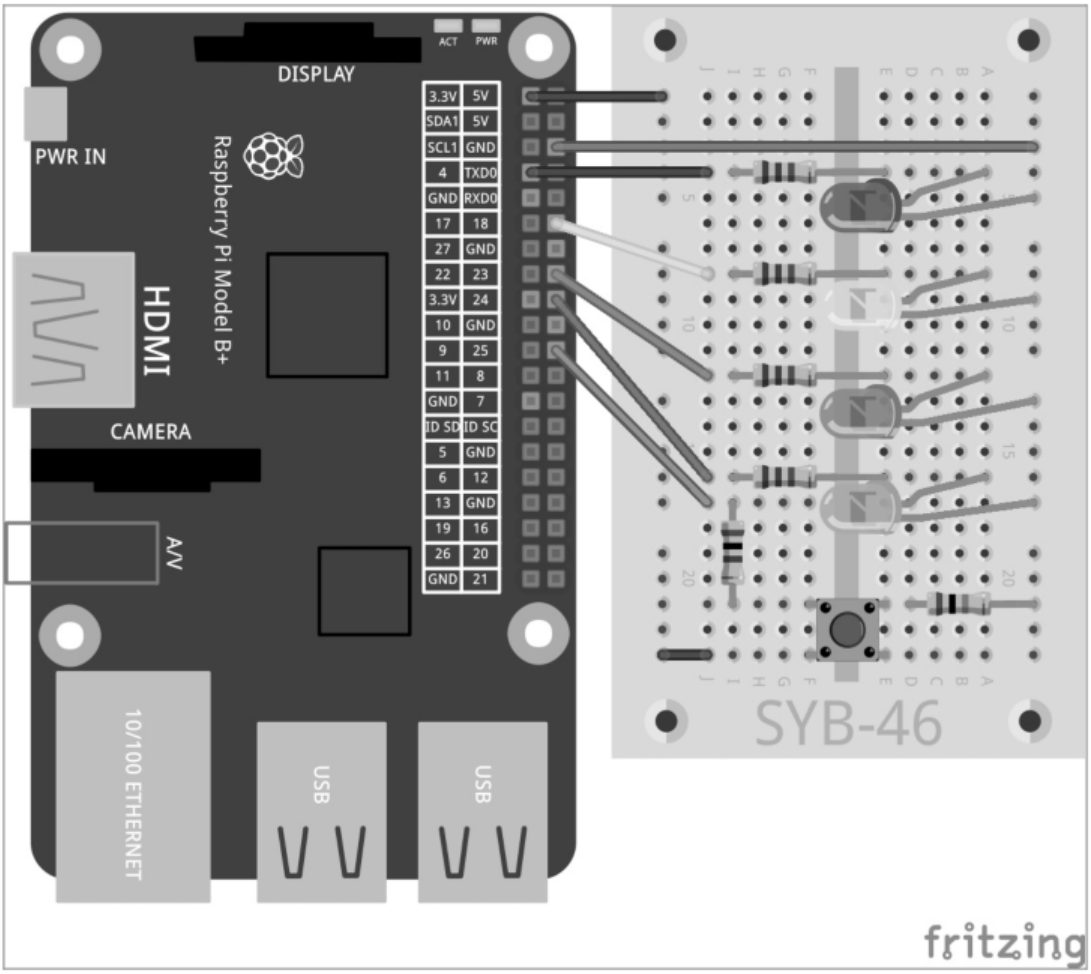
Za naslednji preizkus v vezje vgradite dodatno LED s preduporom tako kot prikazuje slika. Priključite jo na GPIO-vrata 24.



Slika 4.1: Sestav preizkusne ploščice za semafor in utripajočo luč za pešce.

Potrebne komponente:

1 x preizkusna ploščica
1 x rdeča LED
1 x rumena LED
1 x zelena LED
1 x modra LED
4 x 220 Ω upor
5 x priključni kabel



Slika 4.2: Semafor z utripajočo lučjo za pešce.

Program `ampel02.py` krmili nov sistem semaforja. Za razliko od prejšnje različice je program malce nadgrajen.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
rot = 0; gelb = 1; gruen = 2; blau = 3
Ampel=[4,18,23,24]
GPIO.setup(Ampel[rot], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gelb], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gruen], GPIO.OUT, initial=True)
GPIO.setup(Ampel[blau], GPIO.OUT, initial=False)
print ("Ctrl+C zaključi program")
try:
      while True:
            time.sleep(2)
            GPIO.output(Ampel[gruen],False);
            GPIO.output(Ampel[gelb],True)
            time.sleep(0.6)
            GPIO.output(Ampel[gelb],False);
            GPIO.output(Ampel[rot],True)
            time.sleep(0.6)
            for i in range(10):
                  GPIO.output(Ampel[blau],True); time.sleep(0.05)
```

```
                GPIO.output(Ampel[blau],False); time.sleep(0.05)
            time.sleep(0.6)
            GPIO.output(Ampel[gelb],True); time.sleep(0.6)
            GPIO.output(Ampel[rot],False)
            GPIO.output(Ampel[gelb],False)
            GPIO.output(Ampel[gruen],True)
except KeyboardInterrupt:
     GPIO.cleanup()
```

4.1.1 Tako deluje

Potek programa je v veliki meri znan. Med rdečo fazo, ki je sedaj malce daljša, mora modri semafor za pešce hitro utripati.

`blau = 4`
Nova spremenljivka definira LED za semafor za pešce na seznamu.

`Ampel=[4,18,23,24]`
Seznam se poveča na štiri elemente, da je možno krmiljenje štirih LED.

`GPIO.setup(Ampel[blau], GPIO.OUT, initial=False)`
Nova LED se inicializira in je na začetku izklopljena. To je osnovna nastavitev med zeleno fazo semaforja.

```
time.sleep(0.6)
for i in range(10):
     GPIO.output(Ampel[blau],True); time.sleep(0.05)
     GPIO.output(Ampel[blau],False); time.sleep(0.05)
time.sleep(0.6)
```

V ciklu semaforja se 0,6 sekunde po začetku rdeče faze začne zanka, ki skrbi za utripanje modre LED. V ta namen je tukaj uporabljena zanka `for`, ki za razliko od zank `while`, ki so bile uporabljene v prejšnjih preizkusih, vedno uporablja določeno število prehodov zanke in ne poteka tako dolgo, dokler ni izpolnjen določen pogoj za prekinitev.

`for i in range(10):`
Vsaka zanka `for` potrebuje zančni števec – spremenljivko, ki pri vsakem prehodu zanke dobi novo vrednost. Za enostavne zančne števce se je v vseh programskih jezikih uveljavilo ime spremenljivke `i`. Možno je seveda tudi vsako drugo ime. To vrednost lahko tako kot vsako drugo spremenljivko prikličete znotraj zanke, vendar to tukaj ni potrebno. Parameter `range` v zanki navaja, koliko prehodov opravi zanka, natančneje povedano, katere vrednosti lahko zančni števec sprejme. V našem primeru zanka opravi deset prehodov. Zančni števec `i` pri tem dobiva vrednosti od `0` do `9`. Znotraj zanke se vklopi nova modra LED in se po 0,05 sekundah ponovno izklopi. Po nadaljnjih 0,05 sekundah je en prehod zanke zaključen in naslednji se ponovno začne z vklopom LED. Na ta način desetkrat utripne, kar skupno traja 1 sekundo.

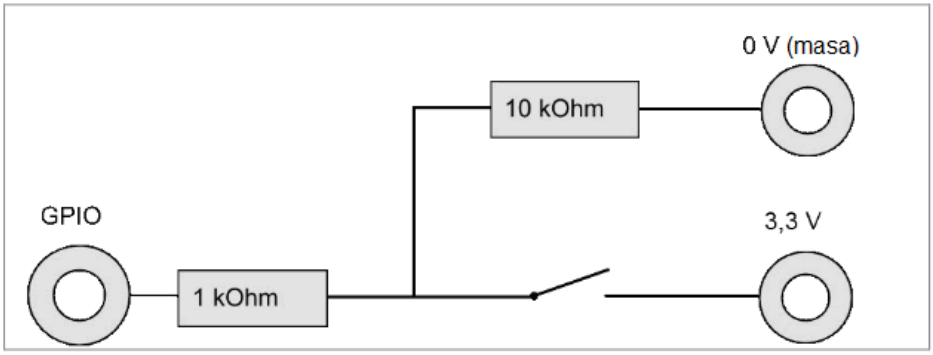
`time.sleep(0,6)`
Z zakasnitvijo 0,6 sekund po zadnjem prehodu zanke se nadaljuje običajen preklopni cikel semaforja, tako da se vklopi rumena LED dodatno k rdeči, ki že sveti. Zaenkrat torej ni veliko novega. Resnično zanimiv postane semafor za pešce, ko ne deluje samodejno, temveč se aktivira šele s pritiskom tipke, tako kot je to značilno za številne semaforje za pešce. V naslednjem preizkusu bo tipkalo, ki je priključeno na GPIO-vrata, simuliralo tipko na resničnem semaforju za pešce.

4.2 Tipkalo na GPIO-priključku

GPIO-vrata se ne uporabljajo samo za izpis podatkov, na primer prek LED, temveč tudi za vnos podatkov. Pri tem morajo biti v programu definirana kot vhod. Za vnos bomo v naslednjem projektu uporabili tipkalo, ki ga nataknete neposredno na preizkusno ploščico. Tipkalo ima štiri priključne pine, pri čemer sta po dva nasproti ležeča (velika razdalja) povezana med seboj. Dokler je tipka pritisnjena, so vsi štirje priključki povezani med seboj. Za razliko od stikala pa tipkalo ne zaskoči. Ko tipko izpustite, se povezava takoj spet prekine. Če se na GPIO-vratih, ki so definirana kot vhod, nahaja +3,3 V signal, se ta interpretirajo kot logična `True` oz. `1`. Teoretično bi lahko torej prek tipkala posamezna GPIO-vrata povezali s +3,3 V priključkom računalnika Raspberry Pi, vendar pa tega nikakor ne smete storiti! GPIO-vrata bi se namreč pri tem preobremenila. Med GPIO-vhod in +3,3 V priključek vedno priključite 1 kΩ zaščitni upor, saj boste tako preprečili, da bi na GPIO-vrata in s tem na procesor steklo preveč toka.

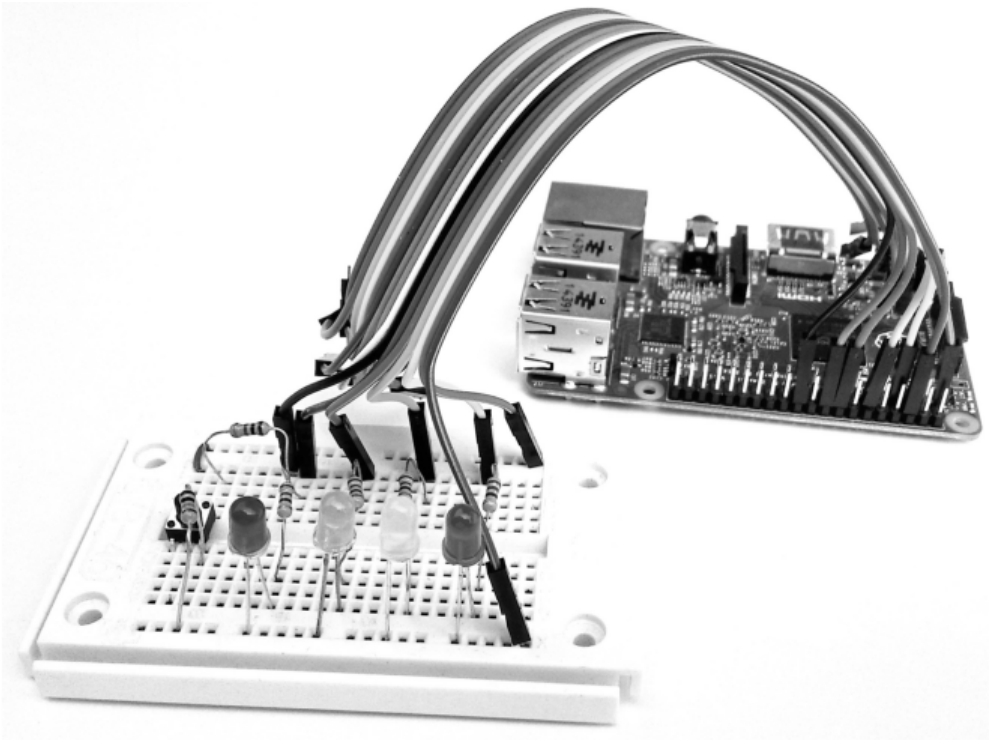
Slika 4.3: Tipkalo z zaščitnim uporom na GPIO-vhodu.

V večini primerov to enostavno vezje že deluje, vendar pa GPIO-vrata pri razklenjenem tipkalu ne bi imela jasno definiranega stanja. Ko program poizveduje po teh vratih, lahko prihaja do naključnih rezultatov. Za preprečitev tega je treba proti masi priključiti primerljivo zelo visok upor – ponavadi je to 10 kΩ. Ta tako imenovani spodnji upor (oz. pull-down upor) status GPIO-vrat pri razklenjenem tipkalu ponovno potegne navzdol na 0 V. Ker je upornost zelo visoka, tudi ne obstaja nevarnost kratkega stika, medtem ko je tipkalo pritisnjeno. V pritisnjenem stanju tipkala sta +3,3 V priključek in vodnik za maso neposredno povezana prek tega upora.



Slika 4.4: Tipkalo z zaščitnim uporom in spodnji upor na GPIO-vhodu.

V skladu z naslednjo sliko vgradite tipkalo z obema uporoma v vezje.



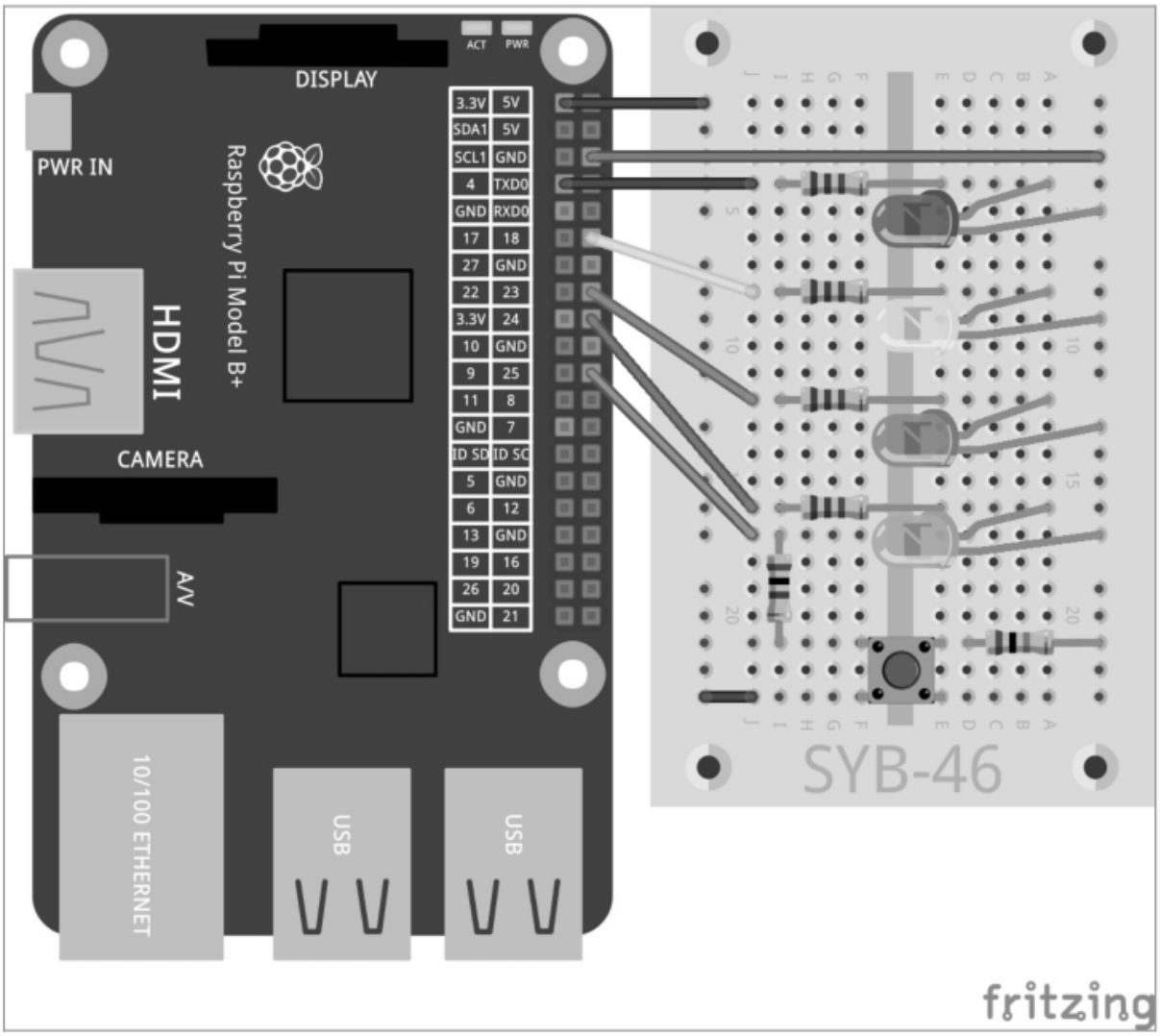
Slika 4.5: Sestav preizkusne ploščice za semafor za pešce s tipko.

Potrebne komponente:

1 x preizkusna ploščica
1 x rdeča LED
1 x rumena LED
1 x zelena LED
1 x modra LED
4 x 220 Ω upor
1 x tipkalo
1 x 1 kΩ upor
1 x 10 kΩ upor
7 x priključni kabel
1 x kratek žični mostiček

Spodnja kontaktna letev tipkala na sliki je prek plus letve preizkusne ploščice povezana s +3,3 V kablom računalnika Raspberry Pi (pin 1). Za povezavo tipkala s plus letvijo smo za ohranjanje preglednosti risbe uporabili kratek žični mostiček. Druga možnost je, da enega izmed spodnjih kontaktov tipkala prek priključnega kabla neposredno povežete s pinom 1 računalnika Raspberry Pi.

Zgornja kontaktna letev tipkala na sliki je prek 1 kΩ zaščitnega upora (rjav-črn-rdeč) povezana z GPIO-vrati 25, prek 10 kΩ spodnjega upora (rjav-črn-oranžen) pa je povezana z vodnikom za maso.



Slika 4.6: Utripajoča luč za pešce s tipkalom.

Program `ampel03.py` krmili nov sistem semaforja s tipkalom za utripajočo luč za pešce.

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

rot = 0; gelb = 1; gruen = 2; blau = 3; taster = 4

Ampel=[4,18,23,24,25]
GPIO.setup(Ampel[rot], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gelb], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gruen], GPIO.OUT, initial=True)
GPIO.setup(Ampel[blau], GPIO.OUT, initial=False)
GPIO.setup(Ampel[taster], GPIO.IN)

print ("pritisnite tipkalo za vklop utripajoče luči za pešce, Ctrl+C
zaključi program")
```

```
try:
      while True:
            if GPIO.input(Ampel[taster])==True:
                  GPIO.output(Ampel[gruen],False)
                  GPIO.output(Ampel[gelb],True)
                  time.sleep(0.6)
                  GPIO.output(Ampel[gelb],False)
                  GPIO.output(Ampel[rot],True)
                  time.sleep(0.6)
                  for i in range(10):
                        GPIO.output(Ampel[blau],True); time.sleep(0.05)
                        GPIO.output(Ampel[blau],False);
                        time.sleep(0.05)
                  time.sleep(0.6)
                  GPIO.output(Ampel[gelb],True)
                  time.sleep(0.6)
                  GPIO.output(Ampel[rot],False);
                  GPIO.output(Ampel[gelb],False)
                  GPIO.output(Ampel[gruen],True); time.sleep(2)
except KeyboardInterrupt:
GPIO.cleanup()
```

4.2.1 Tako deluje

Program smo v primerjavi z zadnjo različico še malo dopolnili.

```
# -*- coding: utf-8 -*-
```

Da so nemški preglasi za `Fußgängerblinklicht` (= utripajoča luč za pešce) pravilno prikazani v izpisu programa – ne glede na to, kako je nastavljen IDLE-vmesnik pri uporabniku – se na začetku definira kodiranje za prikaz posebnih znakov. To vrstico morajo vsebovati vsi programi, ki izpisujejo besedila, v katerih se nahajajo preglasi ali drugi posebni znaki, ki so značilni za posamezne države.

ASCII, ANSI in Unicode

Običajna abeceda ima 26 črk in nekaj preglasov, vse z velikimi in malimi tiskanimi črkami, sem sodi še deset številk in nekaj ločil, kar skupno nanese okoli 100 različnih znakov. Z enim bajtom je možen prikaz 256 različnih znakov. To bi torej moralo zadostovati – tako so mislili na začetku razvoja računalnikov, ko so bile definirane najpomembnejše osnove današnje tehnike.

Kaj kmalu se je izkazalo, da so bili izumitelji nabora znakov ASCII (American Standard Code for Information Interchange), ki temelji na 256 znakih, v zmoti. Američani so bili tisti, ki niso razmišljali izven meja angleškega jezikovnega prostora. V vseh pomembnih svetovnih jezikih, brez vzhodnoazijskih in arabskih jezikov z njihovimi popolnoma svojevrstnimi pisavami, obstaja več sto črk, ki morajo biti prikazane. Na prostih mestih seznama, ki obsega 256 znakov, je prostora samo za nekatere izmed njih.

Ko je bil kasneje vzporedno z naborom znakov ASCII uveden nabor znakov ANSI, ki ga uporabljajo starejše različice operacijskega sistema Windows, so ponovno storili isto napako. Da je bila zmešnjava jezikov popolna, so bili nemški preglasi in druge črke z naglasi razporejene na druga mesta v naboru znakov kot v standardu ASCII.

Za rešitev te težave je bil v 90. letih prejšnjega stoletja uveden Unicode, ki lahko prikazuje vse jezike, tudi egipčanske hieroglife, klinopis in vedski sanskrit – najstarejši ohranjen knjižni jezik na svetu. Najbolj razširjena oblika kodiranja znakov Unicode v čiste besedilne datoteke

je UTF-8. Gre za kodiranje, ki deluje v več računalniških okoljih in se pri prvih 128 znakih popolnoma sklada z ASCII, s čimer je združljivo navzdol s skoraj vsemi sistemi, ki prikazujejo besedila. Kodiranje se vnese v vrstico s komentarjem. Vseh vrstic, ki se začnejo z znakom #, tolmač Python ne interpretira. Kodiranje, ki se mora vedno nahajati čisto na začetku programa, Python Shell da navodila, kako naj prikazuje znake, vendar ne gre za dejansko programsko navodilo. Na ta način lahko v programsko kodo vnašate tudi poljubne lastne komentarje.

Komentarji v programih
Ko pišemo program, kasneje pogosto več ne vemo, kaj smo si pri določenih programskih navodilih milsili. Programiranje je ena izmed najbolj kreativnih dejavnosti, ki jih poznamo, saj brez omejitev, ki jih predstavljajo materiali in orodja, samo iz lastnih idej nekaj ustvarimo. Prav pri programih, ki jih mora tudi druga oseba razumeti ali jih celo nadalje obdelati, so pomembni komentarji. Vzorčni programi ne vsebujejo komentarjev, tako da programska koda ostane pregledna. Vsa programska navodila so izčrpno opisana.

Pri programih, ki jih sami objavljamo, se vedno pojavi vprašanje: Komentarji v maternem jeziku ali v angleščini? Pri komentarjih v maternem jeziku se vsi tujci pritožujejo nad nerazumljivim jezikom, angleških komentarjev tudi sami enkrat več ne razumemo, Britanci pa se posmehujejo slabi angleščini.

```
taster = 4
Ampel=[4,18,23,24,25]
```

V namen enostavnosti smo na seznam za tipkalo dodali dodatni element s številko `4` in GPIO-vrata 25. Na ta način lahko tudi enostavno izberemo druga GPIO-vrata za tipkalo, saj je številka teh vrat, tako kot številka GPIO-vrat pri LED, vnesena samo na tem enem mestu v programu.

```
GPIO.setup(Ampel[taster], GPIO.IN)
```
GPIO-vrata tipkala definiramo kot vhod. Ta definicija prav tako poteka prek `GPIO.setup`, vendar tokrat s parametrom `GPIO.IN`.

```
print ("pritisnite tipkalo za vklop utripajoče luči za pešce, Ctrl+C
zaključi program")
```

Pri zagonu program prikazuje razširjeno sporočilo, ki sporoča, da mora uporabnik pritisniti tipkalo.

```
while True:
     if GPIO.input(Ampel[taster])==True:
```

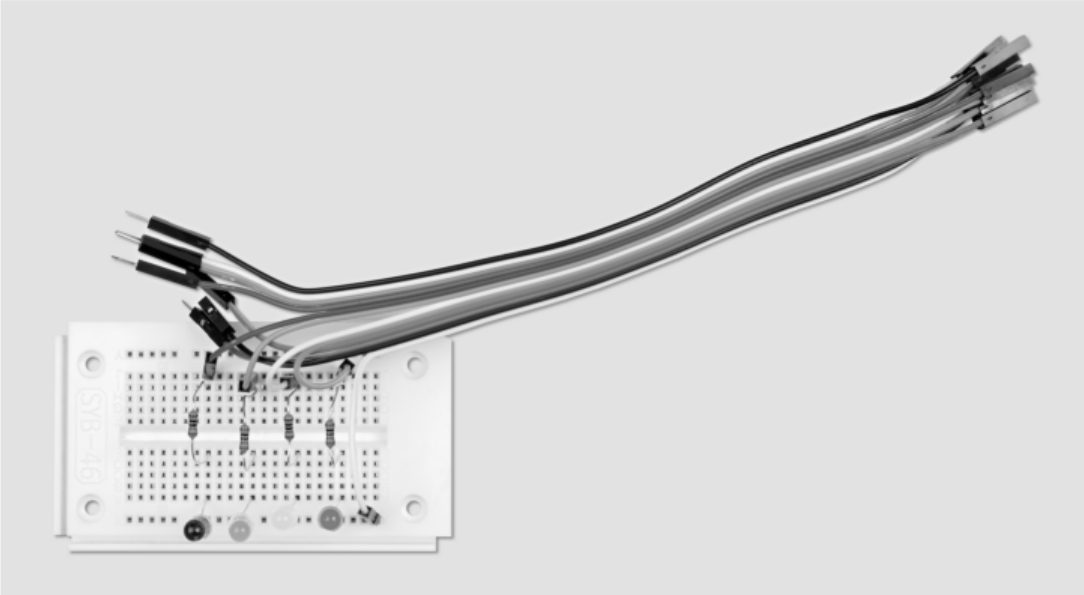
Znotraj neskončne zanke je sedaj vstavljena poizvedba. Naslednja navodila se izvedejo šele takrat, ko GPIO-vrata 25 sprejmejo vrednost `True`, torej ko uporabnik pritisne tipkalo. Do tega trenutka ostane semafor v svoji zeleni fazi. Nadaljnji potek zanke se v bistvu sklada s potekom zadnjega programa. Semafor preklopi prek rumene na rdečo, utripajoča luč desetkrat utripne. Nato semafor spet preklopi iz rdeče prek rumene na zeleno.

```
time.sleep(2)
```
V tem programu obstaja majhna razlika. Zelena faza, ki traja 2 sekundi, je sedaj vstavljena na koncu zanke in ne več na začetku. Kljub temu se uporabi enkrat pri vsakem prehodu zanke, s to razliko, da se cikel semaforja začne takoj in brez zakasnitve, ko pritisnete tipkalo. Za preprečitev, da zelena faza skoraj izpade, ko tipkalo ponovno pritisnete takoj po rumeni fazi, je ta zakasnitev sedaj vstavljena na koncu zanke.

5 Pisani LED-vzorci in tekoče luči

Tekoče luči so vedno znova priljubljeni učinki za vzbujanje pozornosti, pa naj bo to v kleti za zabavo ali pri profesionalnih svetlobnih reklamah. Z Raspberry Pi in nekaj LED je nekaj takega možno enostavno udejanjiti.

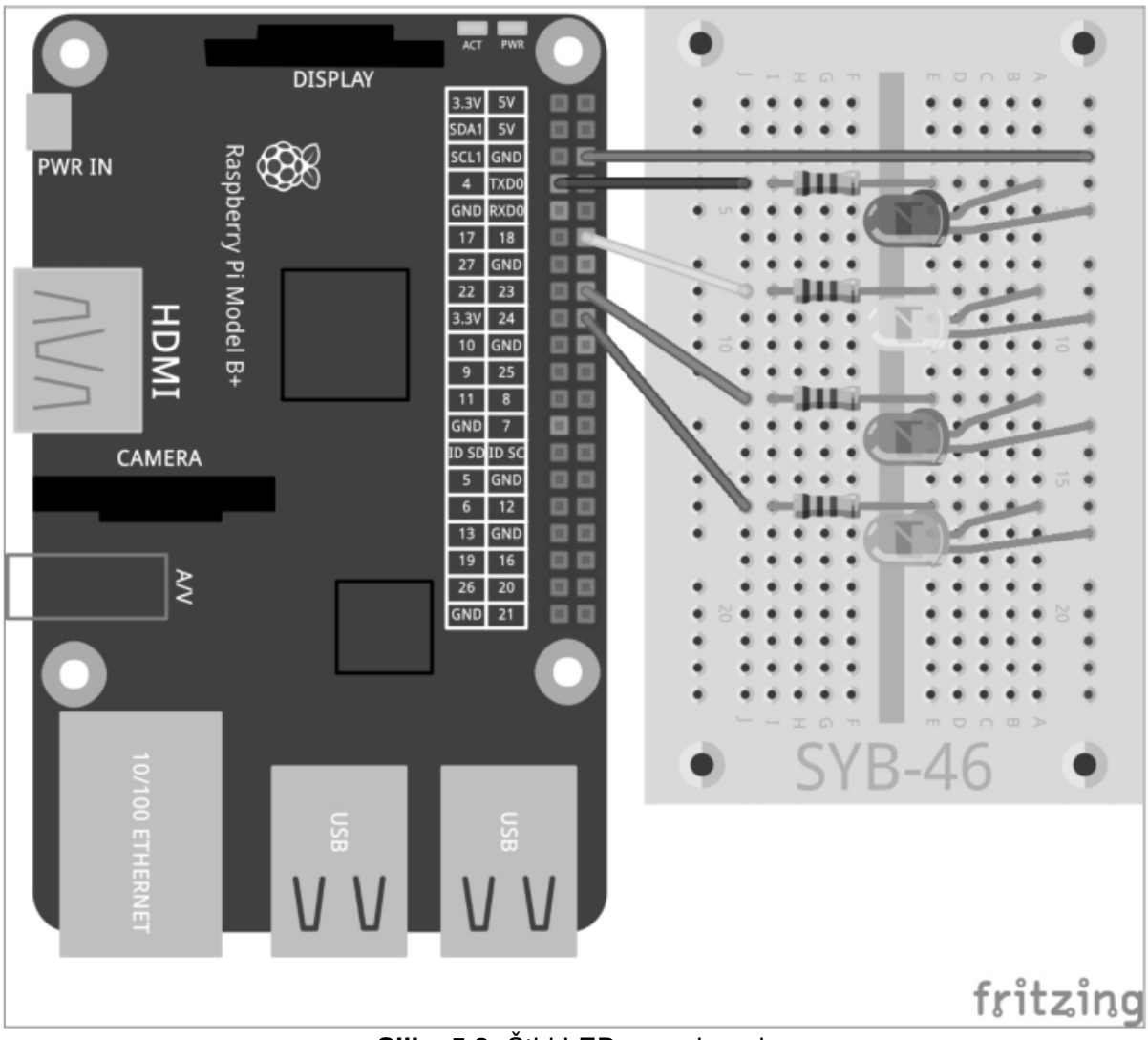
Za naslednji preizkus pritrdite štiri LED s predupori tako kot prikazuje slika. To vezje se sklada s semaforjem za pešce brez tipkala iz prejšnjega preizkusa.



Slika 5.1: Sestav preizkusne ploščice za vzorce in tekoče luči.

Potrebne komponente:

1 x preizkusna ploščica
1 x rdeča LED
1 x rumena LED
1 x zelena LED
1 x modra LED
4 x 220 Ω upor
5 x priključni kabel



Slika 5.2: Štiri LED s predupori.

Na podlagi različnih vzorcev utripanja LED bomo razložili nadaljnje zanke in metode programiranja v Python. Naslednji program nudi različne vzorce LED, ki jih lahko izbere uporabnik z vnosom prek tipkovnice.

Program `ledmuster.py` poskrbi za utripanje LED v različnih vzorcih.

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time
import random

GPIO.setmode(GPIO.BCM)

LED = [4,18,23,24]
for i in LED:
     GPIO.setup(i, GPIO.OUT, initial=0)

z = len(LED); w = 5; t = 0.2

print ("Svetlobni učinki na izbiro");
print ("1 - ciklična tekoča luč")
```

```
print ("2 - tekoča luč naprej in nazaj");
print ("3 - naraščajoča in padajoča")
print ("4 - vse utripajo hkrati");
print ("5 - vse utripajo naključno")
print ("Ctrl+C zaključi program")

try:
     while True:
          e = raw_input ("Prosimo, izberite vzorec: ")
          if e == "1":
               for i in range(w):
                    for j in range(z):
                         GPIO.output(LED[j], True); time.sleep(t)
                         GPIO.output(LED[j], False)
          elif e == "2":
               for i in range(w):
                    for j in range(z):
                         GPIO.output(LED[j], True); time.sleep(t)
                         GPIO.output(LED[j], False)
                    for j in range(z-1, -1, -1):
                         GPIO.output(LED[j], True); time.sleep(t)
                         GPIO.output(LED[j], False)
          elif e == "3":
               for i in range(w):
                    for j in range(z):
                         GPIO.output(LED[j], True); time.sleep(t)
                    time.sleep(2*t)
                    for j in range(z-1, -1, -1):
                         GPIO.output(LED[j], False)
                         time.sleep(t)
                    time.sleep(2*t)
               elif e == "4":
                    for i in range(w):
                         for j in range(z):
                              GPIO.output(LED[j], True)
                    time.sleep(2*t)
                    for j in range(z):
                         GPIO.output(LED[j], False)
                    time.sleep(t)
               elif e == "5":
                    for i in range(w*z):
                         j = random.randint(0,z-1)
                         GPIO.output(LED[j], True); time.sleep(t)
                         GPIO.output(LED[j], False)
               else:
                    print ("Neveljaven vnos")
except KeyboardInterrupt:
     GPIO.cleanup()
```

5.1.1 Tako deluje

Prve vrstice programa z definicijo kodiranja UTF-8 in uvozom potrebnih knjižnic so že znane iz prejšnjih preizkusov. Tukaj se dodatno uvozi knjižnica `random`, ki služi ustvarjanju naključnega vzorca utripanja.

Pri razvoju programa smo pazili na to, da je karseda vsestransko uporaben, torej da ga je možno enostavno nadgraditi na več kot štiri LED. K dobremu slogu programiranja danes sodi takšna fleksibilnost. Na primeru računalnika Raspberry Pi je možno na ta način programirane programe ne samo nadgraditi z novimi GPIO-vrati, temveč jih je možno enostavno predelati na druga GPIO-vrata, če je to potrebno zaradi tehnike strojne opreme.

```
LED = [4,18,23,24]
```
Za LED se ponovno definira seznam s številkami GPIO-vrat, tako da je treba ta vrata fiksno vnesti samo na enem mestu v programu.

```
for i in LED:
     GPIO.setup(i, GPIO.OUT, initial=0)
```

Namesto posamezne inicializacije GPIO-vrat LED, tako kot je to potekalo v prejšnjih programih, tokrat čez seznam `LED` teče zanka `for`. Zančni števec `i` zaporedoma prevzema posamezne vrednosti iz seznama, v našem primeru so to številke GPIO-vrat LED, in več preprosto ne prišteva vrednosti, tako kot v zankah `for`, ki smo jih uporabljali doslej. Na ta način je možna obdelava poljubno dolgih seznamov. Ni nujno, da je dolžina seznama pri razvoju programa znana.

Štiri GPIO-vrata za LED se definirajo kot izhodi in ponastavijo na `0`, da se izklopijo vse LED, ki morda svetijo iz prejšnjih preizkusov.

```
z = len(LED); w = 5; t = 0.2
```

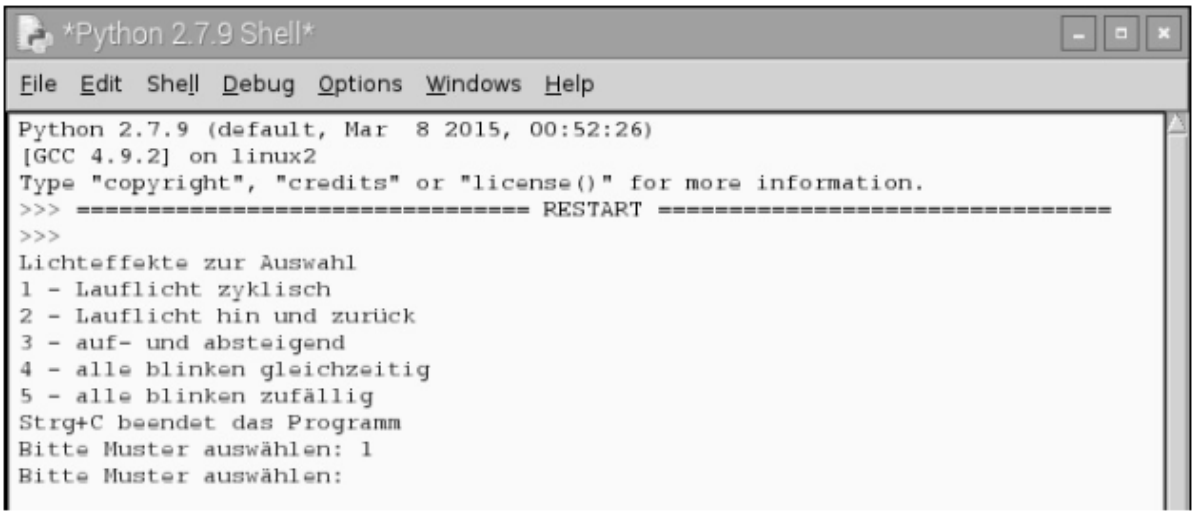
Da program ohrani svojo splošno veljavnost in ga je možno enostavno spreminjati, bomo sedaj definirali še tri spremenljivke:

`z`	Število LED	Število LED se s pomočjo funkcije `len()` samodejno prevzame iz seznama `LED`.
`w`	Ponovitve	Vsak vzorec se v standardnih nastavitvah petkrat ponovi, da ga je možno bolje prepoznati. To število lahko poljubno spreminjate in nato velja za vse vzorce.
`t`	Čas	Ta spremenljivka navaja, kako dolgo je LED pri utripanju vklopljena. Premor, ki temu sledi, traja enako dolgo. Ime `t` se je v skoraj vseh programskih jezikih uveljavilo za spremenljivke, ki shranjujejo časovne intervale v programih.

Vrednosti, ki so določene kot spremenljivke, so samo na tem mestu fiksno vnesene v programu, tako da jih je možno enostavno spremeniti. Po teh definicijah se zažene dejanski program.

```
print ("Svetlobni učinki na izbiro");
print ("1 – ciklična tekoča luč")
print ("2 – tekoča luč naprej in nazaj");
print ("3 – naraščajoča in padajoča")
print ("4 – vse utripajo hkrati");
print ("5 – vse utripajo naključno");
print ("Ctrl+C zaključi program")
```

Te vrstice uporabniku nudijo navodila na zaslonu glede tega, s katero številsko tipko je prikazan kateri vzorec.



Slika 5.3: Program na zaslonu.

Po prikazu izbire se začne glavna zanka programa. Pri tem tudi tukaj uporabimo neskončno zanko `while True`:, ki je vključena v navodilu `try...except`.

`e = raw_input ("Prosimo, izberite vzorec: ")`
Takoj na začetku zanke program čaka na vnos uporabnika, ki se shrani v spremenljivki `e`. Funkcija `raw_input()` prevzame vnos v besedilni obliki in ga ne interpretira. Ravno nasprotno pa se z `input()` vnesene matematične operacije ali imena spremenljivk neposredno interpretirajo. V večini primerov je torej `raw_input()` boljša izbira, saj se vam tukaj ni treba brigati za številne eventualnosti možnih vnosov.

Program čaka, dokler uporabnik ne vnese črke in pritisne tipke `[Enter]`. V odvisnosti od tega, katero število je uporabnik vnesel, se mora sedaj s pomočjo LED prikazati določen vzorec. Za poizvedbo tega uporabimo konstrukt `if...elif...else`.

Vzorec št. 1

Če je bil vnos `1`, potem se izvede del programa, ki je zamaknjen za to vrstico.

`if e == "1":`
Upoštevajte, da zamiki v Python ne služijo samo optiki. Tako kot pri zankah se tudi takšne poizvedbe uvedejo z zamikom.

Enako ni enako enako
Python uporablja dve vrsti enačajev. Enostavni `=` se uporablja za dodeljevanje določene vrednosti spremenljivki. Dvojni enačaj `==` se uporablja v poizvedbah in preverja, če sta dve vrednosti resnično enaki.

Če torej uporabnik prek tipkovnice vnese `1`, se začne zanka, ki ustvarja ciklično tekočo luč. Te zanke so za vse uporabljene LED-vzorce načeloma enako sestavljene.

`for i in range(w):`
Zunanja zanka tako dolgo ponavlja vzorec, kot je navedeno v spremenljivki `w`, ki ste jo na začetku definirali. V tej zanki se nahaja dodatna, ki ustvarja posamezni vzorec. Ta se pri vsakem vzorcu razlikuje.

```
for j in range(z):
      GPIO.output(LED[j], True); time.sleep(t)
      GPIO.output(LED[j], False)
```

V primeru enostavne ciklične tekoče luči ta zanka zaporedoma steče enkrat za vsako LED na seznamu. Koliko LED to je, ste shranili na začetku programa v spremenljivki `z`. Vklopi se LED s številko trenutnega stanja zančnega števca. Nato program počaka, da preteče čas, ki ste ga shranili na začetku v spremenljivki `t`, nato pa ponovno izklopi LED. Takoj nato se začne naslednji prehod zanke z naslednjo LED. Zunanja zanka petkrat ponovi celotno notranjo zanko.

Vzorec št. 2

Ko uporabnik vnese številko `2`, se začne podobna zanka. Pri tem pa se LED ne preštevajo samo v eni smeri, temveč se na koncu tekoče luči ponovno vrnejo v obratnem vrstnem redu. Luč teče izmenično naprej in nazaj.

```
elif e == "2":
```

Nadaljnje poizvedbe po prvi uporabljajo poizvedbo `elif`, kar pomeni, da se izvedejo samo takrat, ko je prejšnja poizvedba vrnila rezultat `False`.

```
for i in range(w):
      for j in range(z):
            GPIO.output(LED[j], True); time.sleep(t)
            GPIO.output(LED[j], False)
      for j in range(z-1, -1, -1):
            GPIO.output(LED[j], True); time.sleep(t)
            GPIO.output(LED[j], False)
```

Tudi tukaj se uporabljajo zanke, ki se nahajajo ena v drugi. Po prvi notranji zanki, ki se sklada s prej opisanim delom programa, torej potem ko sveti LED s številko 3, se začne dodatna zanka za tekočo luč v nasprotni smeri. Elementi s seznama so vedno oštevilčeni tako, da se začnejo z 0. Četrta LED ima torej številko 3.

Da zanka teče nazaj, bomo uporabili nadgrajeno sintakso `for...range()`. Namesto vnosa samo ene končne vrednosti je možno navesti tudi tri parametre: začetna vrednost, vrednost koraka in končna vrednost. V našem primeru so to:

Začetna vrednost	z-1	Spremenljivka `z` vsebuje število LED. Ker se oštevilčenje elementov na seznamu začne z `0`, ima zadnja LED število `z-1`.
Vrednost koraka	-1	Pri vrednosti koraka `-1` vsak prehod zanke šteje eno število nazaj.
Končna vrednost	-1	Končna vrednost v zanki je vedno prva vrednost, ki ni dosežena. Pri prvi zanki, ki šteje naprej, se zančni števec začne pri `0` in v našem primeru doseže vrednosti `0`, `1`, `2`, `3` za naslavljanje LED. Vrednost `4` pri štirikratnem prehodu zanke ni dosežena. Zanka, ki šteje nazaj, se mora končati z `0` in tako ne sme doseči vrednosti `-1` kot prve vrednosti.

Opisana druga zanka poskrbi, da štiri LED zaporedoma utripajo v obratni smeri. Nato zunanja zanka začne celoten cikel na novo, ki v tem primeru traja dvakrat tako dolgo kot v prvem delu programa, saj vsaka LED dvakrat utripa.

Vzorec št. 3

Ko uporabnik vnese številko `3`, se začne podobna zanka. Tukaj se LED prav tako preštevajo v obeh smereh, vendar se ne izklopijo takoj po vklopu.

```
elif e == "3":
     for i in range(w):
          for j in range(z):
               GPIO.output(LED[j], True); time.sleep(t)
          time.sleep(2*t)
          for j in range(z-1, -1, -1):
               GPIO.output(LED[j], False); time.sleep(t)
          time.sleep(2*t)
```

Prva notranja zanka vklopi LED eno za drugo s časovno zakasnitvijo. Na koncu zanke, ki ga prepoznate po zamiku vrstice `time.sleep(2*t)`, se počaka dvojni čas zakasnitve. Tako dolgo vse LED svetijo. Nato se začne naslednja zanka, ki šteje nazaj in ponovno izklopi eno LED za drugo. Tudi tukaj se na koncu, ko so vse LED izklopljene, počaka dvojni čas zakasnitve, nato pa zunanja zanka ponovno začne celoten cikel.

Vzorec št. 4

Ko uporabnik vnese številko `4`, se začne drugi vzorec utripanja, pri katerem vse LED hkrati utripajo in se ne vklapljajo ena za drugo.

```
elif e == "4":
     for i in range(w):
          for j in range(z):
               GPIO.output(LED[j], True)
          time.sleep(2*t)
          for j in range(z):
               GPIO.output(LED[j], False)
          time.sleep(t)
```

Ker več GPIO-vrat ni možno naenkrat vklopiti ali izklopiti z enim samim navodilom, se tudi tukaj uporabljajo zanke, vendar brez časovne zakasnitve znotraj zanke. Štiri LED se vklopijo neposredno ena za drugo. Človeško oko to zaznava kot hkratni pojav. Na koncu prve notranje zanke program počaka dvojni čas zakasnitve, nato pa ponovno izklopi vse LED.

Z različnimi časi vklopa (svetenje) in izklopa je možno pri utripajočih lučeh ustvariti različne učinke. Utripanje zaznamo takrat, ko je čas vklopa (svetenje) daljši od časa izklopa. Zelo kratki časi vklopa pri primerljivo dolgih časih izklopa ustvarjajo učinek bliskovne luči.

Vzorec št. 5

Ko uporabnik vnese `5`, LED utripajo popolnoma naključno.

```
elif e == "5":
     for i in range(w*z):
          j = random.randint(0,z-1)
          GPIO.output(LED[j], True); time.sleep(t)
          GPIO.output(LED[j], False)
```

Ker se tukaj ne uporabljajo zanke, ki se nahajajo ena v drugi, poskrbimo za to, da zanka pogosteje teče. Z množenjem spremenljivk `w` in `z` vsaka LED utripa v povprečju tako pogosto kot v prvem vzorcu.

Funkcija `random.randint()` zapiše naključno število v spremenljivko `j`. To naključno število je večje ali enako prvemu parametru in manjše ali enako drugemu parametru, tako da lahko v našem primeru prevzame vrednosti `0`, `1`, `2`, `3`.

Naključno izbrana LED se vklopi in po preteku časa zakasnitve ponovno izklopi. Nato se zanka na novo začne in naključno se izbere nova LED.

Neveljaven vnos

Pri vseh programih, ki zahtevajo vnose s strani uporabnikov, je treba prestreči napačne vnose. Če uporabnik vnese nekaj nepredvidljivega, se mora program odzvati na to.

```
else:
     print ("Neveljaven vnos")
```

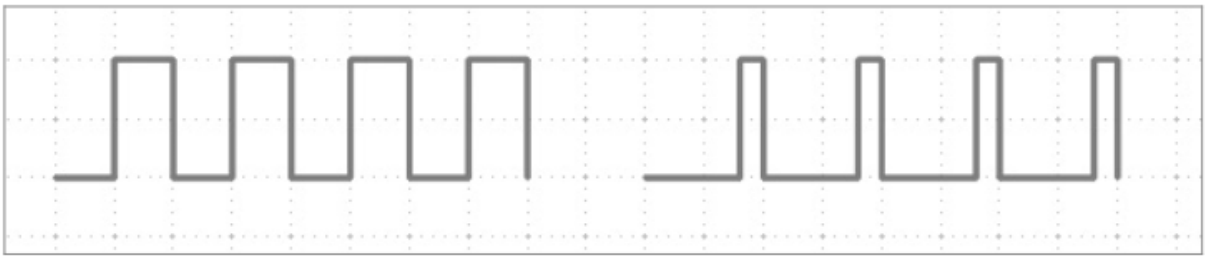
Ko uporabnik vnese karkoli drugega, se izvede navodilo, ki je navedeno pod `else`. Ta del poizvedbe vedno velja takrat, ko nobena od drugih poizvedb ne prinese pravega rezultata. V našem primeru program prikaže sporočilo na zaslonu.

Tako kot v prejšnjih preizkusih se program zaključi prek `KeyboardInterrupt`, tako da uporabnik pritisne kombinacijo tipk `[Ctrl]+[C]`. Zadnja vrstica zapre uporabljena GPIO-vrata in s tem izklopi vse LED.

6 Zatemnjevanje LED s pulznoširinsko modulacijo

LED so značilne komponente za izhodne signale v digitalni elektroniki. Nahajajo se lahko v dveh različnih stanjih, vklop in izklop, 0 in 1 ali `True` in `False`. Isto velja za GPIO-vrata, ki so definirana kot digitalni izhodi. Potemtakem zatemnjevanje LED teoretično ne bi bilo možno.

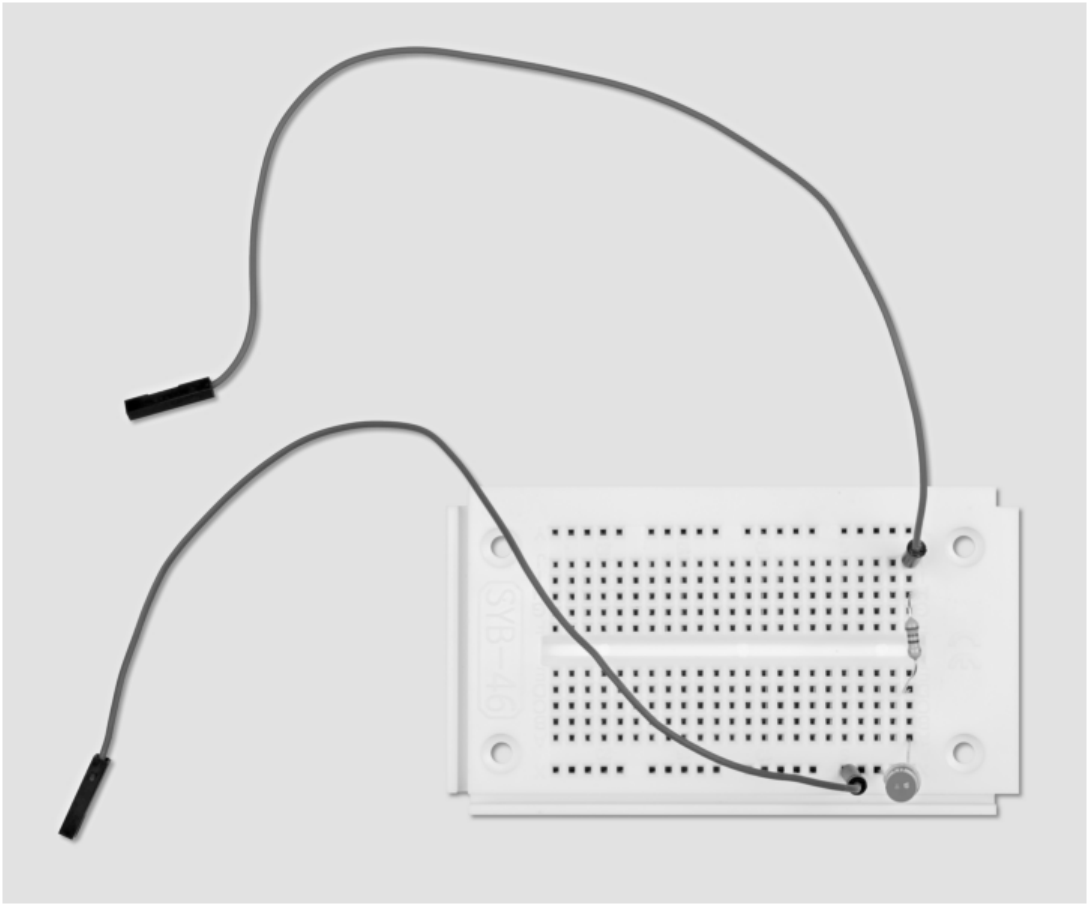
Z uporabo trika pa je kljub temu možna regulacija svetilnosti LED na digitalnih GPIO-vratih. Če poskrbimo za dovolj hitro utripanje LED, potem človeško oko tega več ne zaznava kot utripanje. Tehnika, ki ji pravimo pulznoširinska modulacija, ustvarja pulzirajoč signal, ki se vklaplja in izklaplja v zelo kratkih intervalih. Napetost signala ostaja vedno ista, spreminja se samo razmerje med nivojem `False` (0 V) in nivojem `True` (+3,3 V). Delovni cikel navaja razmerje med dolžino vklopljenega stanja in skupnim trajanjem preklopnega cikla.



Slika 6.1: Levo: delovni cikel 50 % – desno: delovni cikel 20 %.

Manjši kot je delovni cikel, krajši je čas svetenja LED znotraj enega preklopnega cikla. S tem ta LED deluje temnejša od LED, ki je trajno vklopljena.

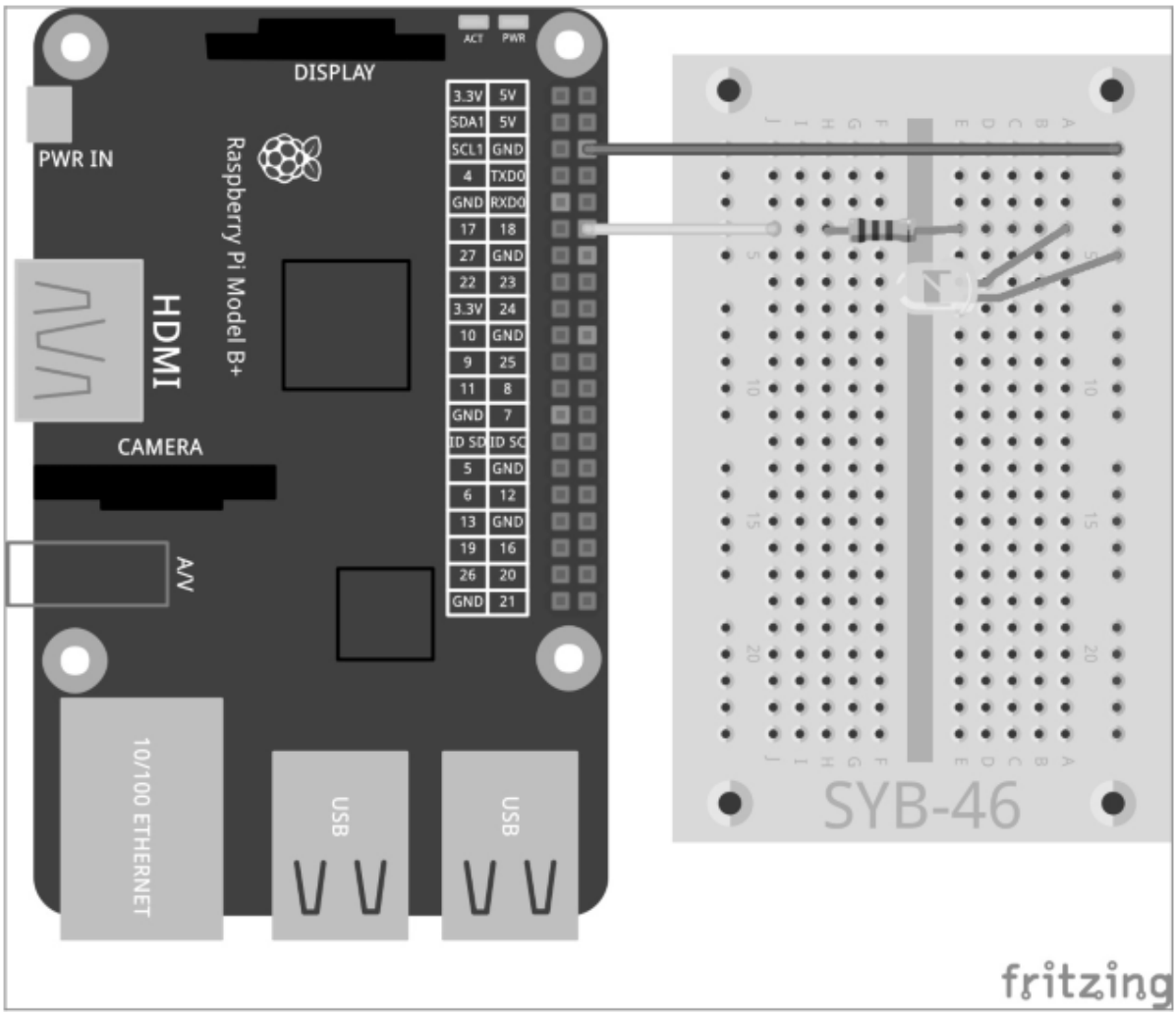
Za naslednji preizkus priključite LED prek predupora na GPIO-vrata 18.



Slika 6.2: Sestav preizkusne ploščice z LED.

Potrebne komponente:

1 x preizkusna ploščica
1 x rumena LED
1 x 220 Ω upor
2 x priključni kabel



Slika 6.3: LED na GPIO-vratih 18.

Program `leddimmen01.py` ciklično zatemnjuje LED svetleje in temneje ter pri tem uporablja lastno funkcijo pulznoširinske modulacije (PWM) iz GPIO-knjižnice. PWM-signal (signal pulznoširinske modulacije) se generira kot lastna nit. Na ta način lahko zatemnjeno LED uporabljate v programu (skoraj) tako kot LED, ki običajno sveti.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM); LED = 18
GPIO.setup(LED, GPIO.OUT)
print ("Ctrl+C zaključi program")
p = GPIO.PWM(LED, 50); p.start(0)
try:
     while True:
          for c in range(0, 101, 2):
               p.ChangeDutyCycle(c); time.sleep(0.1)
          for c in range(100, -1, -2):
               p.ChangeDutyCycle(c); time.sleep(0.1)
except KeyboardInterrupt:
     p.stop(); GPIO.cleanup()
```

6.1.1 Tako deluje

Del tega programa se vam bo zdel znan, nekateri elementi pa vam bodo popolnoma neznani, saj na tem mestu naredimo ekskurz v programiranje, ki je orientirano na objekt. Na začetku se uvozijo knjižnice, tako kot vam je že znano. Tokrat se za GPIO-vrata 18 določi ena sama spremenljivka `LED`, ki se inicializira kot izhod.

```
print ("Ctrl+C zaključi program")
```

Ker tudi ta program deluje s konstruktom `try...except` in ga mora ustaviti uporabnik, je prikazana ustrezna informacija za uporabnika.

```
p = GPIO.PWM(LED, 50)
```

Funkcija `GPIO.PWM()` iz GPIO-knjižnice je odločilna za izhodne PWM-signale. Ta funkcija potrebuje dva parametra: GPIO-vrata in frekvenco PWM-signala. V našem primeru se GPIO-vrata določijo s spremenljivko `LED`, frekvenca pa je 50 Hz (nihaji na sekundo).

Zakaj je 50 Hz idealna frekvenca za pulznoširinsko modulacijo
Spremembe svetlobe, ki so hitrejše od 20 Hz, človeško oko več ne zaznava. Ker električno omrežje z izmenično napetostjo v Evropi uporablja frekvenco 50 Hz, številna svetlobna telesa utripajo s to frekvenco, ki jo oko ne zazna. Če bi LED utripala z več kot 20 Hz, vendar manj kot 50 Hz, bi lahko prihajalo do motenj z drugimi viri svetlobe, pri čemer učinek zatemnitve ne bi več deloval enakomerno.

`GPIO.PWM()` ustvarja tako imenovani objekt, ki je shranjen v spremenljivki `p`. Takšni objekti so veliko več kot samo enostavne spremenljivke. Objekti lahko imajo različne lastnosti, nanje pa je možno vplivati s tako imenovanimi metodami. Metode so ločene s piko, ki je navedena neposredno za imenom objekta.

```
p.start(0)
```

Metoda `start()` začne generacijo PWM-signala. Pri tem je treba navesti še delovni cikel. V našem primeru je delovni cikel `0`, LED je torej vedno izklopljena. Sedaj se začne neskončna zanka, v kateri se neposredno ena za drugo nahajata dve zanki, ki skrbita, da LED izmenično postaja svetlejša in temnejša.

```
for c in range(0, 101, 2):
```

Zanka šteje v korakih po `2` od `0` do `100`. Kot konec zanke `for` je vedno navedena vrednost, ki ravno ni dosežena, v našem primeru `101`.

```
p.ChangeDutyCycle(c)
```

V vsakem prehodu zanke metoda `ChangeDutyCycle()` nastavi delovni cikel PWM-objekta na vrednost zančnega števca, torej vsakič za 2 % višje, dokler se pri zadnjem prehodu ne nahaja na 100 % in LED ne sveti s svojo polno svetilnostjo.

```
time.sleep(0.1)
```

Pri vsakem prehodu zanke znaša čakalni čas 0,1 sekunde, dokler naslednji prehod ponovno ne poviša delovnega cikla za 2 %.

```
for c in range(100, -1, -2):
     p.ChangeDutyCycle(c); time.sleep(0.1)
```

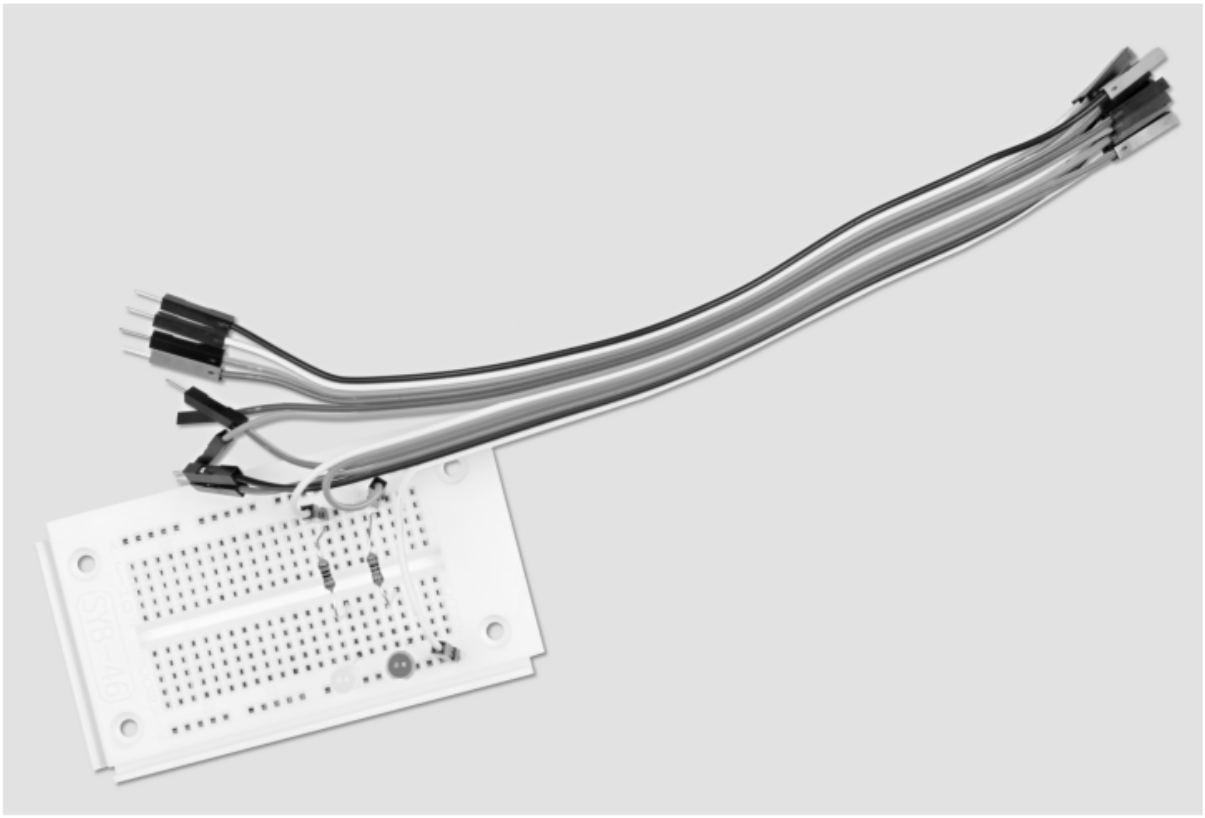
Potem ko LED doseže polno svetilnost, jo druga zanka v skladu z isto shemo ponovno zmanjšuje. Ta zanka šteje od `100` navzdol v korakih po `-2`. Ta cikel se ponavlja, dokler uporabnik ne pritisne kombinacije tipk `[Ctrl]+[C]`.

```
except KeyboardInterrupt:
     p.stop(); GPIO.cleanup()
```

`KeyboardInterrupt` sedaj dodatno sproži metodo `stop()` pri PWM-objektu. Ta metoda zaključi ustvarjanje PWM-signala. Nato se GPIO-vrata ponastavijo, tako kot vam je že znano iz prejšnjih programov.

6.1.2 Neodvisno zatemnjevanje dveh LED

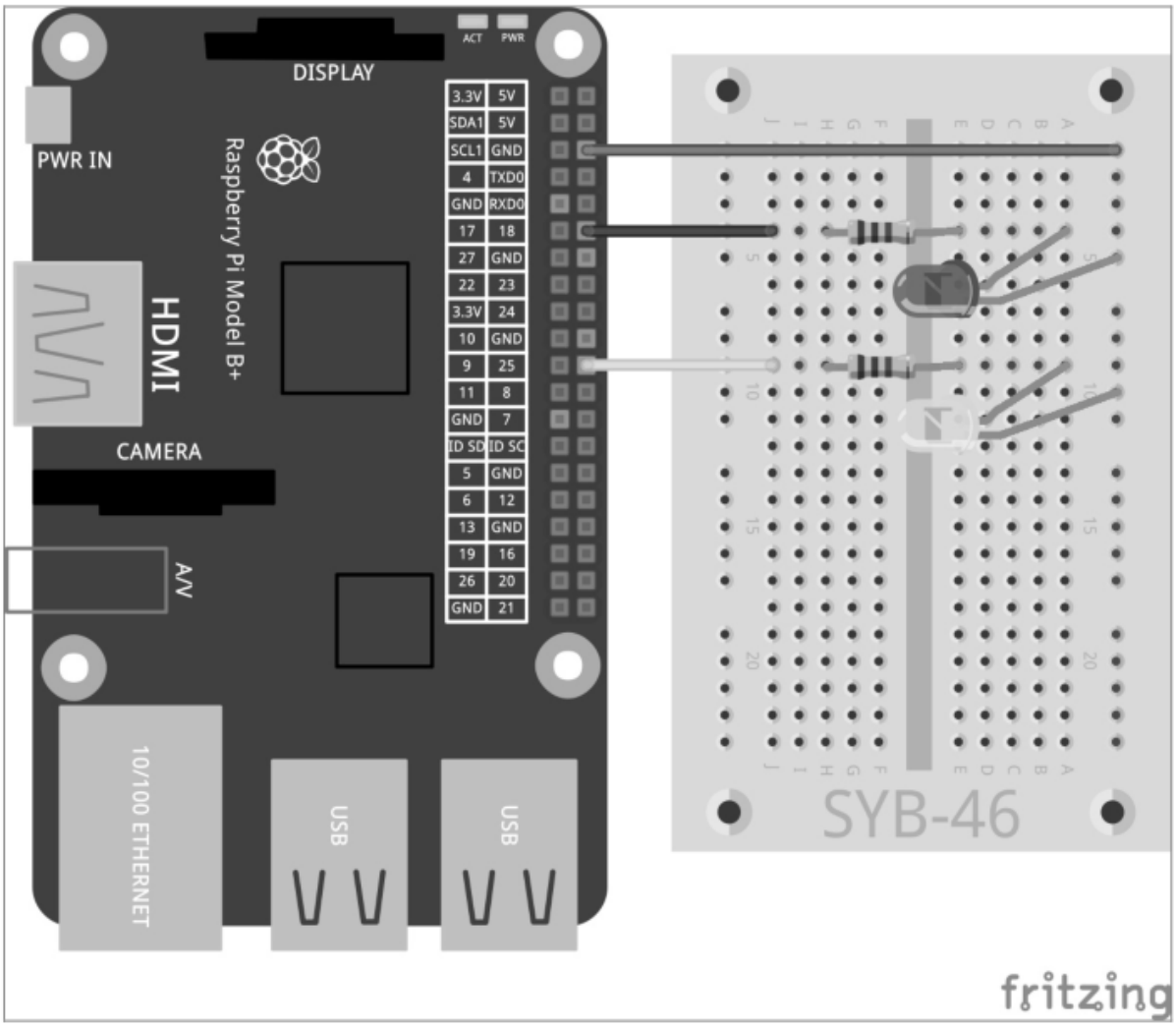
Ker za programiranje PWM-signala ni potreben programski čas v skripti Python, je možno tudi zatemnjevanje več LED neodvisno druga od druge, tako kot ponazarja tudi naslednji preizkus. Pri tem priključite dodatno LED prek predupora na GPIO-vrata 25.



Slika 6.4: Sestav preizkusne ploščice za zatemnjevanje dveh LED.

Potrebne komponente:

1 x preizkusna ploščica
1 x rumena LED
1 x rdeča LED
2 x 220 Ω upor
3 x priključni kabel



Slika 6.5: Druga LED na GPIO-vratih 25.

Program `leddimmen02.py` eno LED ciklično zatemnjuje svetleje in temneje, medtem ko druga LED sicer skupaj s prvo LED postane svetlejša, vendar v drugem ciklu ne postane temnejša, temveč ponovno postaja svetlejša od 0 naprej in pri tem hitro utripa.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM); LED = [18,25]
GPIO.setup(LED[0], GPIO.OUT); GPIO.setup(LED[1], GPIO.OUT)

print ("Ctrl+C zaključi program")

p = GPIO.PWM(LED[0], 50); q = GPIO.PWM(LED[1], 50)
p.start(0)
q.start(0)

try:
      while True:
            for c in range(0, 101, 2):
                  p.ChangeDutyCycle(c); q.ChangeDutyCycle(c)
                  time.sleep(0.1)
            q.ChangeFrequency(10)
```

```
            for c in range(0, 101, 2):
                 p.ChangeDutyCycle(100-c); q.ChangeDutyCycle(c)
                 time.sleep(0.1)
            q.ChangeFrequency(50)

except KeyboardInterrupt:
     p.stop(); GPIO.cleanup()
```

6.1.3 Tako deluje

Osnovna struktura programa se sklada s strukturo prejšnjega preizkusa z nekaj majhnimi nadgradnjami.

```
LED = [18,25]; GPIO.setup(LED[0], GPIO.OUT); GPIO.setup(LED[1], GPIO.OUT)
```

Namesto spremenljivke za GPIO-vrata se sedaj definira seznam z dvema spremenljivkama, s čimer se dvojna GPIO-vrata, `18` in `25`, inicializirajo kot izhoda za LED.

```
p = GPIO.PWM(LED[0], 50); q = GPIO.PWM(LED[1], 50); p.start(0); q.start(0)
```

Nato se ustvarita tudi dva objekta `p` in `q`, ki ustvarjata PWM-signale za obe LED, pri čemer ima vsak signal po 50 Hz.

```
for c in range(0, 101, 2):
     p.ChangeDutyCycle(c); q.ChangeDutyCycle(c)
     time.sleep(0.1)
```

V prvi zanki se delovna cikla obeh PWM-objektov hkrati povečujeta korak za korakom. Obe LED se v tej fazi enako vedeta.

```
q.ChangeFrequency(10)
```

Na koncu te zanke, ko obe LED dosežeta svojo polno svetilnost, se frekvenca PWM-signala druge LED prek metode `ChangeFrequency()` zmanjša na 10 Hz. Oko to frekvenco še zaznava kot utripanje.

```
for c in range(0, 101, 2):
     p.ChangeDutyCycle(100-c); q.ChangeDutyCycle(c)

     time.sleep(0.1)
```

Sedaj se začne druga zanka, zaradi preglednosti tudi tokrat z naraščajočim štetjem. Za prvo LED iz PWM-objekta `p`, ki se mora v tem ciklu korak za korakom zatemnjevati, se v vsakem prehodu izračunajo ustrezne vrednosti za delovni cikel. Pri drugi LED iz PWM-objekta `q` se delovni cikel enostavno spet prišteva. Učinek utripanja nastane zaradi spremenjene frekvence.

```
q.ChangeFrequency(50)
```

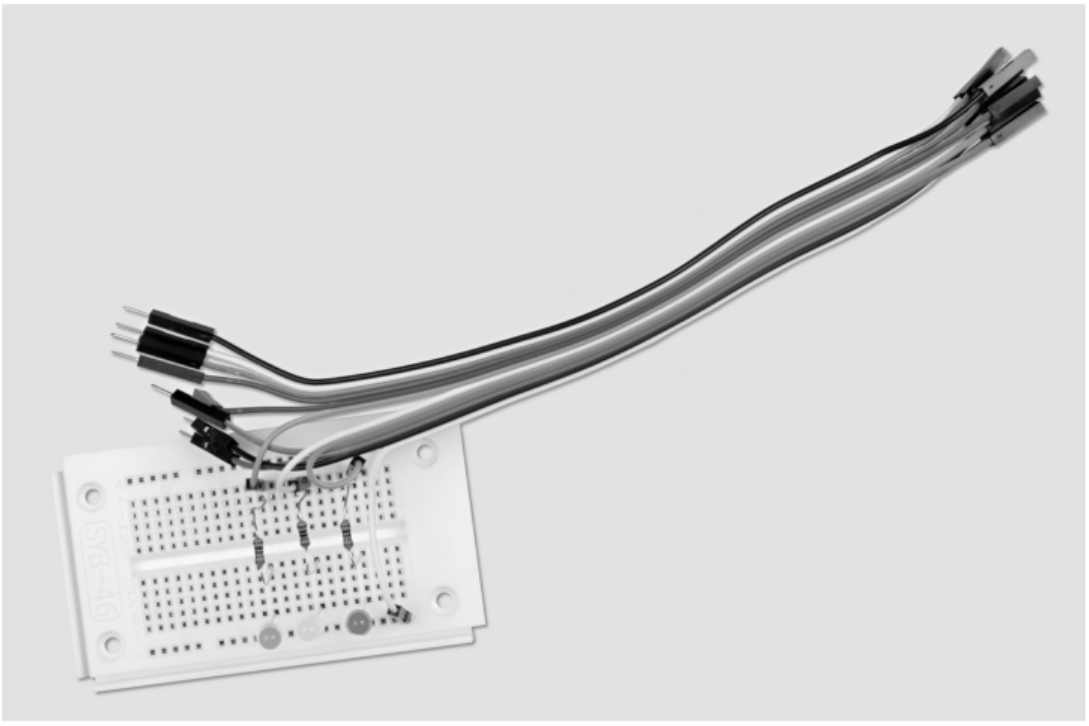
Na koncu druge zanke se frekvenca te LED ponovno ponastavi na 50 Hz, tako da lahko v naslednjem ciklu spet počasi postaja svetlejša točno tako kot prva LED.

7 Prikaz zasedenosti spominske kartice z LED

Spominske kartice so tako kot trdi diski vedno prehitro polne. Pri tem si marsikdo želi enostaven optični prikaz zasedenosti, tako da se vedno na prvi pogled vidi, kdaj je pomnilniški prostor že skoraj popolnoma zaseden. S tremi LED je to možno na Raspberry Pi zelo udobno udejanjiti. Pri tem se uporabljajo funkcije operacijskega sistema, katerih poizvedba poteka prek Python.

Slika 7.1: Seveda je možen prikaz nezasedenega pomnilniškega prostora tudi neposredno v upravitelju datotek na Raspberry Pi.

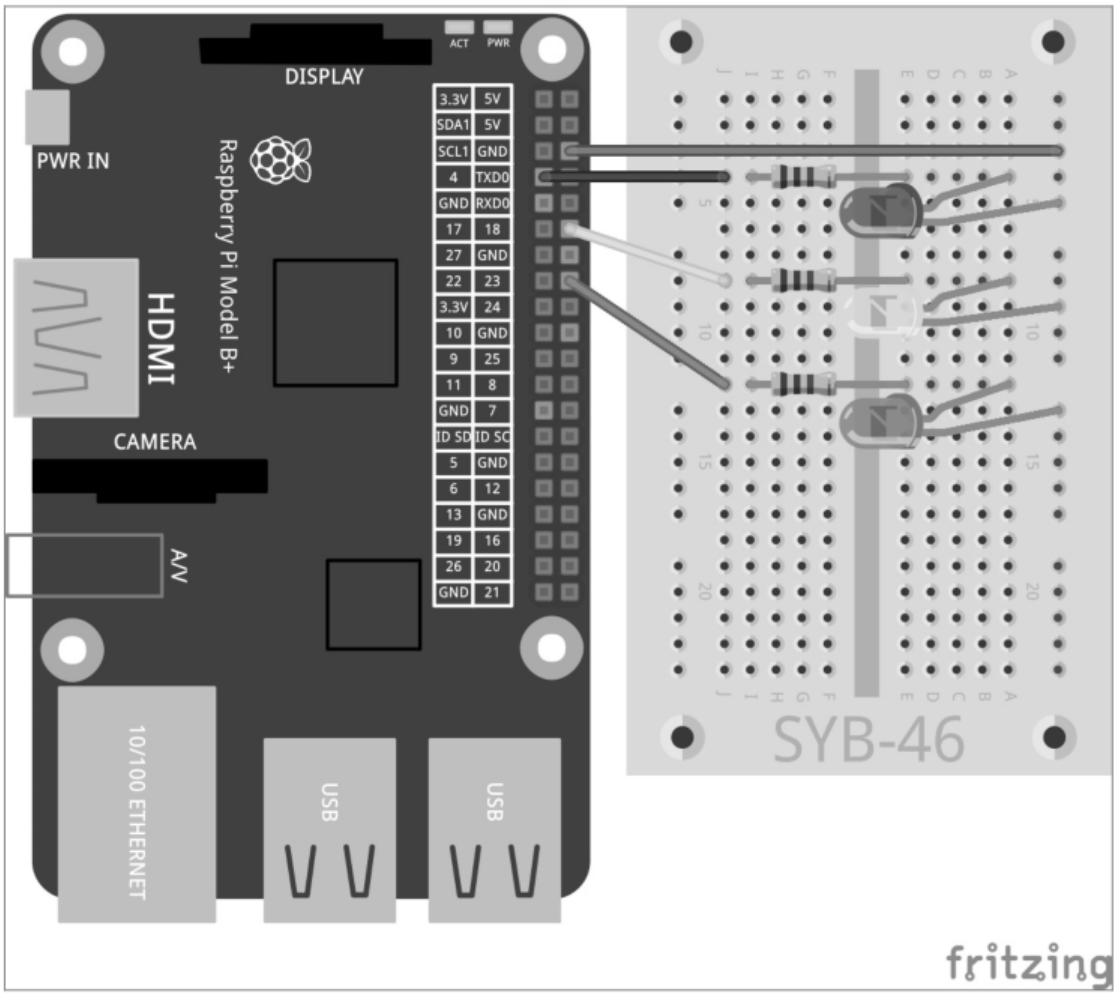
Za prikaz nezasedenega pomnilniškega prostora bomo uporabili tri LED iz vezja s semaforjem, ki svetijo v različnih barvnih kombinacijah.



Slika 7.2: Sestav preizkusne ploščice za prikaz zasedenosti spominske kartice.

Potrebne komponente:

1 x preizkusna ploščica
1 x rdeča LED
1 x rumena LED
1 x zelena LED
3 x 220 Ω upor
4 x priključni kabel



Slika 7.3: Tri LED prikazujejo nezaseden pomnilniški prostor na spominski kartici.

Program `speicheranzeige.py` v odvisnosti od nezasedenega pomnilniškega prostora na spominski kartici nudi različne LED-prikaze:

Nezaseden pomnilniški prostor	LED-prikaz
< 1 MB	Rdeč
1 MB do 10 MB	Rdeče-rumen
10 MB do 100 MB	Rumen
100 MB do 500 MB	Rumeno-zelen
> 500 MB	Zelen

Tabela 7.1: Tako je prikazana zasedenost spominske kartice.

```
import RPi.GPIO as GPIO
import time
import os

g1 = 1; g2 = 10; g3 = 100; g4 = 500

GPIO.setmode(GPIO.BCM)
LED = [4,18,23]
for i in range(3):
     GPIO.setup(LED[i], GPIO.OUT, initial=False)

print ("Strg+C beendet das Programm")
```

```
try:
      while True :
            s = os.statvfs('/')
            f = s.f_bsize * s.f_bavail / 1000000
            if f < g1:
                  x = "100"
            elif f < g2:
                  x = "110"
            elif f < g3:
                  x = "010"
            elif f < g4:
                  x = "011"
            else:
                  x = "001"

            for i in range(3):
                  GPIO.output(LED[i], int(x[i]))
            time.sleep(1.0)

except KeyboardInterrupt:
      GPIO.cleanup()
```

Ko pustite, da program deluje, LED ves čas prikazujejo nezaseden pomnilniški prostor na spominski kartici. Preizkusite, kako deluje, tako da prek omrežja kopirate velike datoteke na spominsko kartico in jih spet izbrišete. Prikaz se samodejno posodablja.

7.1.1 Tako deluje

Za izračun nezasedenega pomnilniškega prostora program uporablja modul Python `os`, ki nudi osnovne funkcije operacijskega sistema.

`import os`
Modul `os` je treba tako kot vse druge module uvoziti na začetku programa.

`g1 = 1; g2 = 10; g3 = 100; g4 = 500`
Te vrstice definirajo meje območij za nezaseden pomnilniški prostor, na katerih naj prikaz preklopi. Zaradi enostavnosti program uporaba megabajte in ne bajte, saj si je možno ta števila bolje predstavljati. Meje lahko kadarkoli drugače določite, vse štiri vrednosti morajo biti samo razporejene v naraščajoči velikosti.

```
GPIO.setmode(GPIO.BCM)
LED = [4,18,23]
for i in range(3):
      GPIO.setup(LED[i], GPIO.OUT, initial=False)
```

Seznam definira številke GPIO-vrat treh LED. Nato zanka inicializira tri GPIO-vrata kot izhode in vse LED preklopi v izklopljeno stanje.

Tudi v tem preizkusu bomo uporabili konstrukt `try...except` in neskončno zanko, da program samodejno vedno znova deluje, dokler ga uporabnik ne prekine s `[Ctrl]+[C]`. Nato sledijo dejanske zanimive funkcije, ki dostopajo do operacijskega sistema in poizvedujejo o nezasedenem pomnilniškem prostoru.

`s = os.statvfs('/')`

Statistični modul `os.statvfs()` iz knjižnice `os` nudi različne statistične informacije o datotečnem sistemu. Te informacije se znotraj neskončne zanke pri vsakem prehodu zanke na novo zapišejo kot objekt v spremenljivko `s`.

```
f = s.f_bsize * s.f_bavail / 1048576
```

Sedaj metoda `s.f_bsize` nudi velikost pomnilniškega bloka v bajtih,. `s.f_bavail` pa navaja število nezasedenih blokov. Proizvod iz teh dveh vrednosti nato navaja število nezasedenih bajtov, ki se pri tem deli z 1.048.576, rezultat pa je število nezasedenih megabajtov. Rezultat se shrani v spremenljivki `f`.

```
if f < g1:
     x = "100"
```

Če je nezaseden pomnilniški prostor manjši od prve mejne vrednosti (1 MB), se zaporedje znakov `x`, ki navaja vzorec vklopljenih LED, nastavi na `"100"`. Prva, rdeča LED mora svetiti. Vzorec je enostavna veriga znakov iz številk 0 in 1.

```
elif f < g2:
     x = "110"
elif f < g3:
     x = "010"
elif f < g4:
     x = "011"
```

S pomočjo poizvedb `elif` poizvedujete po nadaljnjih mejnih vrednostih, v skladu s tem pa se nastavijo LED-vzorci, ko prvo vprašanje ne drži, torej ko je na voljo več kot 1 MB nezasedenega pomnilniškega prostora.

```
else:
     x = "001"
```

Če nobena izmed poizvedb ne drži, torej je na voljo več nezasedenega pomnilniškega prostora, kot navaja najvišja mejna vrednost, se LED-vzorec nastavi na `"001"`. Zadnja, zelena LED mora svetiti.

```
for i in range(3):
     GPIO.output(LED[i], int(x[i]))
```

Zanka določa vrednosti na GPIO-izhodih za tri LED. Vse LED zaporedoma dobijo dodeljeno številsko vrednost posamezne številke iz zaporedja znakov, `0` ali `1`. Za deaktivacijo ali aktivacijo GPIO-izhodov se lahko uporabljajo vrednosti `0` in `1` prav tako kot `False` in `True`. Funkcija `int()` iz znaka izračuna njegovo številsko vrednost. Zančni števec `i` prebere znak iz določenega položaja vzorčne verige znakov.

```
time.sleep(1.0)
```

Program čaka 1 sekundo do naslednjega prehoda zanke. Za varčevanje z zmogljivostjo lahko določite tudi daljše čakalne čase, po poteku katerih se naj izračun nezasedenega pomnilniškega prostora ponovi.

Na tem mestu se zanka `while...True` začne od začetka. Če uporabnik vmes pritisne kombinacijo tipk `[Ctrl]+[C]`, se sproži `KeyboardInterrupt` in zapusti zanko. Nato se GPIO-vrata zaprejo, s čimer se LED izklopijo.

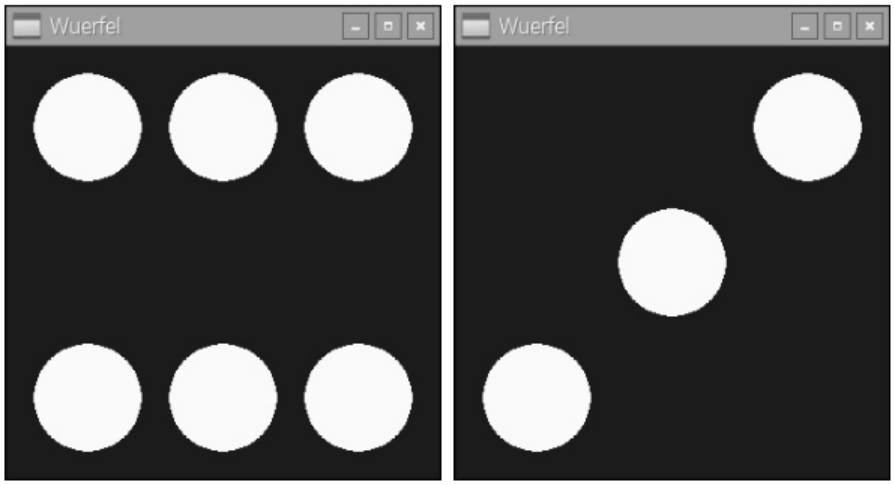
8 Grafična igralna kocka

Zanimiva igra potrebuje grafiko in ne samo besedilnega prikaza kot v časih prvih DOS-računalnikov.

Knjižnica PyGame nudi predhodno definirane funkcije in objekte za prikaz grafike in programiranje iger.

Tako ni potrebe, da bi vse izumljali od začetka.

Za številne igre je potrebna kocka, vendar pa pogosto ni nobena pri roki. Naslednji primer programa prikazuje, kako enostavna je uporaba računalnika Raspberry Pi kot kocke s pomočjo Python in PyGame.



Slika 8.1: Raspberry Pi kot kocka.

Upravljanje kocke mora biti karseda enostavno in potekati s samo eno tipko, naključi rezultat metanja kocke pa mora biti grafično prikazan kot "prava" kocka. Naslednji program `wuerfel.py` simulira takšno kocko na zaslonu.

```
# -*- coding: utf-8 -*-
import pygame, sys, random
from pygame.locals import *
pygame.init()

FELD = pygame.display.set_mode((320, 320))
pygame.display.set_caption("Wuerfel")

BLAU = (0, 0, 255); WEISS = (255, 255, 255)
P1 = ((160, 160)); P2 = ((60, 60)); P3 = ((160, 60));
P4 = ((260, 60))
P5 = ((60, 260)); P6 = ((160, 260)); P7 = ((260, 260))
mainloop = True

print "Za metanje kocke pritisnite poljubno tipko, z [Esc]
zaključite z igro"
```

```
while mainloop:
     for event in pygame.event.get():
          if event.type == QUIT or (event.type == KEYUP and
     event.key == K_ESCAPE):
               mainloop = False
          if event.type == KEYDOWN:
               FELD.fill(BLAU)
               ZAHL = random.randrange (1, 7); print ZAHL
               if ZAHL == 1:
                    pygame.draw.circle(FELD, WEISS, P1, 40)
               if ZAHL == 2:
                    pygame.draw.circle(FELD, WEISS, P2, 40)
                    pygame.draw.circle(FELD, WEISS, P7, 40)
               if ZAHL == 3:
                    pygame.draw.circle(FELD, WEISS, P1, 40)
                    pygame.draw.circle(FELD, WEISS, P4, 40)
                    pygame.draw.circle(FELD, WEISS, P5, 40)
               if ZAHL == 4:
                    pygame.draw.circle(FELD, WEISS, P2, 40)
                    pygame.draw.circle(FELD, WEISS, P4, 40)
                    pygame.draw.circle(FELD, WEISS, P5, 40)
                    pygame.draw.circle(FELD, WEISS, P7, 40)
               if ZAHL == 5:
                    pygame.draw.circle(FELD, WEISS, P1, 40)
                    pygame.draw.circle(FELD, WEISS, P2, 40)
                    pygame.draw.circle(FELD, WEISS, P4, 40)
                    pygame.draw.circle(FELD, WEISS, P5, 40)
                    pygame.draw.circle(FELD, WEISS, P7, 40)
               if ZAHL == 6:
                    pygame.draw.circle(FELD, WEISS, P2, 40)
                    pygame.draw.circle(FELD, WEISS, P3, 40)
                    pygame.draw.circle(FELD, WEISS, P4, 40)
                    pygame.draw.circle(FELD, WEISS, P5, 40)
                    pygame.draw.circle(FELD, WEISS, P6, 40)
                    pygame.draw.circle(FELD, WEISS, P7, 40)
     pygame.display.update()
pygame.quit()
```

8.1.1 Tako deluje

Ta program prikazuje številne nove funkcije, predvsem za prikaz grafike s knjižnico PyGame, ki je primerna ne samo za prikaz grafike v igrah temveč tudi za vsako drugo grafiko na zaslonu.

```
import pygame, sys, random
from pygame.locals import *
pygame.init()
```

Te tri programske vrstice se nahajajo na začetku skoraj vsakega programa, ki uporablja PyGame. Poleg že omenjenega modula `random` za ustvarjanje naključnih števil se naložita sam modul `pygame` kot tudi modul `sys`, ki vsebuje pomembne sistemske funkcije, ki jih potrebuje PyGame, kot je npr. odpiranje in zapiranje oken. Uvozijo se vse funkcije iz knjižnice PyGame, nato pa se inicializira dejanski modul PyGame.

```
FELD = pygame.display.set_mode((320, 320))
```

Ta pomembna funkcija v vsakem programu, ki uporablja grafični izhod, definira risalno površino, tako imenovani surface, ki ima v našem primeru velikost 320 x 320 slikovnih točk in dobi ime FELD. Upoštevajte način pisanja v dvojnih oklepajih, ki se v splošnem uporablja za grafične koordinate zaslona. Takšni surface je prikazan v novem oknu na zaslonu.

```
pygame.display.set_caption("Wuerfel")
```

Ta vrstica vnese ime okna.

```
BLAU = (0, 0, 255); WEISS = (255, 255, 255)
```

Ti vrstici definirata uporabljeni barvi: modra in bela. V program bi lahko tudi vsakič neposredno vnesli vrednosti barv, kar pa ne bi ravno doprineslo k preglednosti.

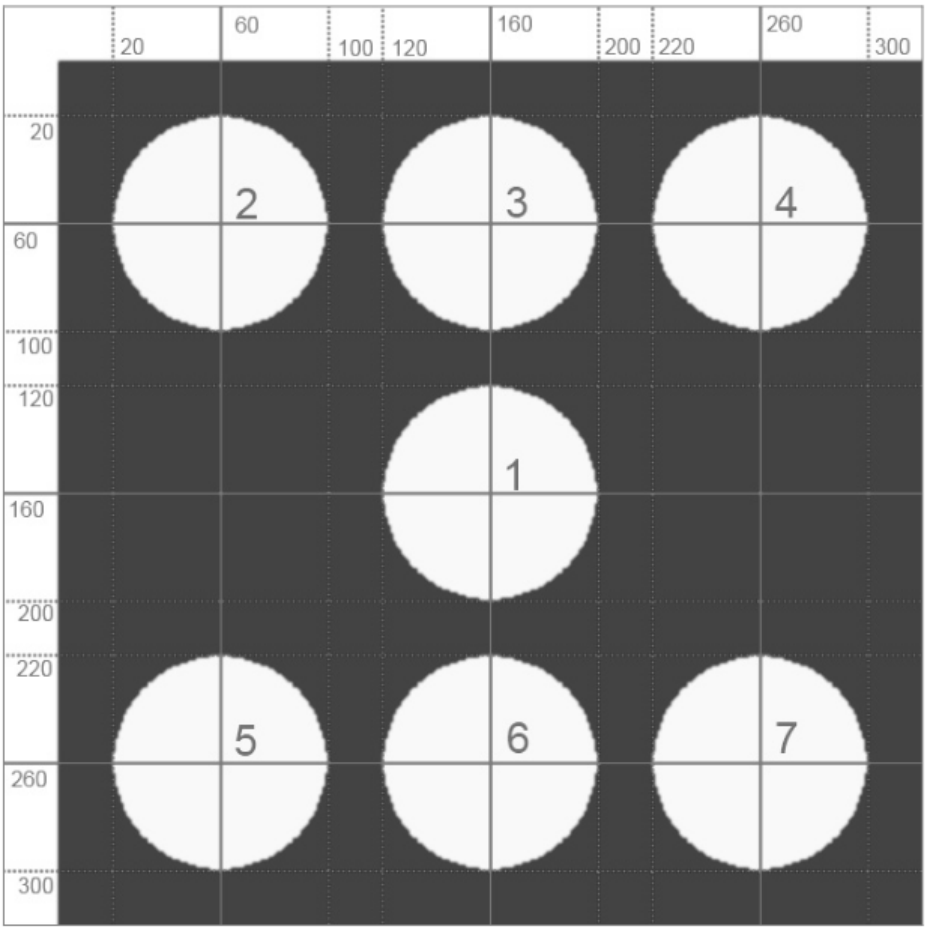
Prikaz barv na zaslonu
Tako kot v večini drugih programskih jezikov so barve v Python definirane s tremi števili med 0 in 255, ki določajo tri deleže barv rdeča, zelena in modra. Zasloni uporabljajo seštevalno mešanje barv, pri katerem vsi trije deleži barv v polni nasičenosti skupaj dosežejo belo barvo.

```
P1 = ((160, 160)); P2 = ((60, 60)); P3 = ((160, 60)); P4 = ((260,
60)); P5 = ((60, 260)); P6 = ((160, 260)); P7 = ((260, 260))
```

Te vrstice določajo središča pik na kocki. Na risalnem polju v velikosti 320 x 320 slikovnih točk se tri osi pik kocke nahajajo na koordinatah `60`, `160` in `260`.

Koordinatni sistem za računalniško grafiko
Vsaka točka v oknu oz. na objektu na risalni površini je definirana s koordinato x in y. Ničelna točka koordinatnega sistema ni levo spodaj, tako kot se učimo v šoli, temveč levo zgoraj. Tako kot besedilo, ki ga beremo od levo zgoraj v smeri desno spodaj, os x sega od leve proti desni, os y pa od zgoraj navzdol.

Sedem točk `P1` do `P7` se nanaša na središča pik kocke, ki so navedena v grafiki. Vsaka pika kocke ima polmer 40 slikovnih točk. Pri medosni razdalji 80 slikovnih točk ostane torej 20 slikovnih točk med pikami kocke in 20 slikovnih točk med robovi okna.



Slika 8.2: Pike kock in njihove koordinate.

Na tem mestu se skupaj z drugimi spremenljivkami nastavi na `True` tudi spremenljivka po imenu `mainloop`, ki jo boste kasneje potrebovali za glavno zanko igre.

```
mainloop = True
```

S tem so ustvarjene osnove in dejanska igra se lahko začne.

```
print "Za metanje kocke pritisnite poljubno tipko, z [Esc] zaključite z igro"
```

Ta vrstica uporabniku na kratko razloži, kaj mora storiti. Z vsakim pritiskom poljubne tipke na tipkovnici na novo vržete kocko. `print` vedno piše v okno Python Shell in ne v novo grafično okno.

```
while mainloop:
```

Sedaj se začne glavna zanka igre. V številnih igrah se uporablja neskončna zanka, ki se vedno znova ponavlja in nenehno poizveduje o kakšnih aktivnostih uporabnika. Nekje v zanki je treba definirati pogoj za prekinitev, ki poskrbi za to, da je možno zaključiti z igro.

V ta namen se tukaj uporablja spremenljivka `mainloop`, ki sprejme samo obe logični vrednosti `True` in `False` (prav in narobe, vklop in izklop). Na začetku se nahaja na `True` in se poizveduje pri vsakem prehodu zanke. Če v času trajanja zanke sprejme vrednost `False`, se zanka zaključi pred naslednjim prehodom.

```
for event in pygame.event.get():
```

Ta vrstica prebere zadnjo aktivnost uporabnika in jo shrani kot `event`. V igri sta na voljo samo dve vrsti aktivnosti uporabnika, ki sta relevantni za igro: Uporabnik pritisne eno tipko in z njo meče kocko ali pa uporabnik želi zaključiti z igro.

```
if event.type == QUIT or (event.type == KEYUP and event.key ==
K_ESCAPE):
     mainloop = False
```

Obstajata dve možnosti za zaključitev igre: Lahko kliknete na simbol x v zgornjem desnem robu okna ali pa pritisnete tipko `[Esc]`. Ko kliknete na simbol x, gre za `event.type == QUIT`, ki ga nudi operacijski sistem. Ko pritisnete poljubno tipko in jo spet izpustite, je `event.type == KEYUP`. V tem primeru se pritisnjena tipka dodatno shrani v `event.key`.

Opisana poizvedba `if` preveri, če želi uporabnik zapreti okno ali (`or`) pa je pritisnil in izpustil poljubno tipko in (`and`) je to tipka z interno oznako `K_ESCAPE`. Če je temu tako, se spremenljivka `mainloop` prestavi na `False`, kar zaključi glavno zanko igre pred naslednjim prehodom.

`if event.type == KEYDOWN:`
Druga vrsta aktivnosti uporabnika, ki se med igro vedno znova pojavlja in se ne pojavi samo enkrat, je pritisk poljubne tipke s strani uporabnika. Pri tem ni pomembno, katera je ta tipka (izjema je tipka [`Esc`]). Ko pritisnete poljubno tipko (`KEYDOWN`), aktivirate pomemben del programa, ki ustvari in tudi prikaže rezultat metanja kocke.

`FELD.fill(BLAU)`
Najprej se objekt na risalni površini z oznako `FELD`, dejansko programsko okno, napolni z barvo, ki je bila na začetku definirana kot `BLAU`, da prekrije prejšnji rezultat metanja kocke.

```
ZAHL = random.randrange (1, 7)
```

Sedaj naključna funkcija `random` generira naključno število med `1` in `6` ter ga shrani v spremenljivki `ZAHL`.

```
print ZAHL
```
Ta vrstica zapiše rezultat metanja kocke v okno Python Shell, ki služi samo kontroli. Če ne potrebujete izpisa podatkov na besedilni osnovi, lahko to vrstico tudi izpustite.

```
if ZAHL == 1:
     pygame.draw.circle(FELD, WEISS, P1, 40)
```

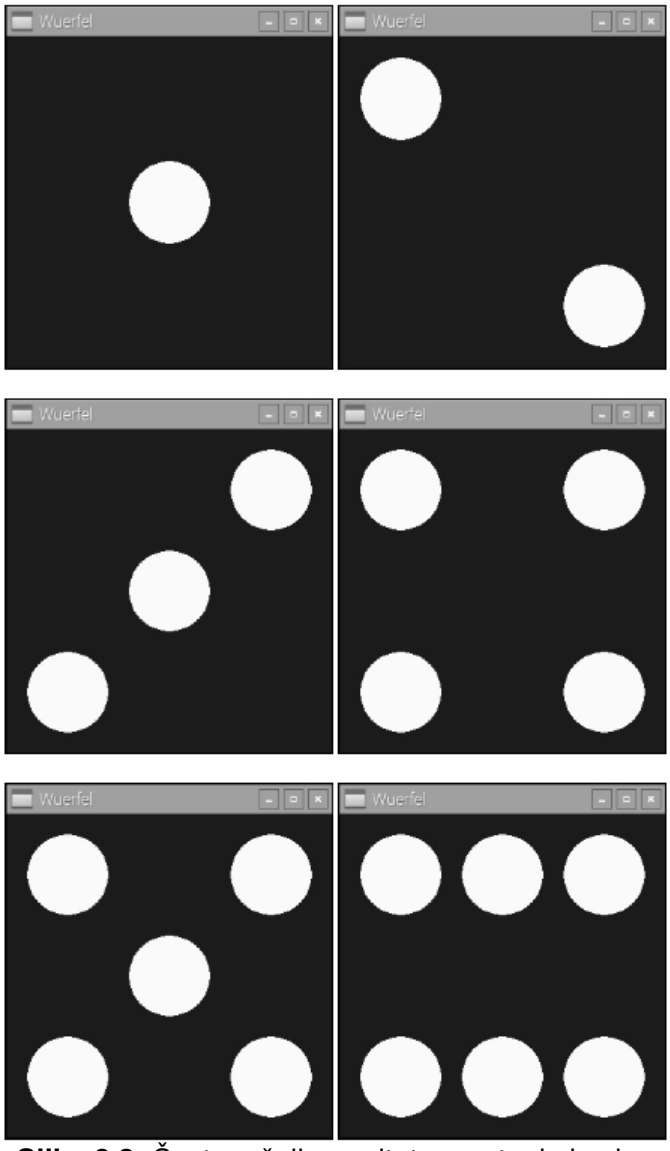
Nato sledi šest poizvedb, ki vse potekajo po isti shemi. Ko ima naključno število pri metanju kocke določeno vrednost, se v skladu s tem prikaže ena do šest pik kocke. Funkcija `pygame.draw.circle(`, ki se uporablja za to, potrebuje štiri ali pet parametrov:

- *Površina* nudi podatek o risalni površini, na kateri se riše, v našem primeru `FELD`.
- *Barva* navaja barvo kroga, v našem primeru je to predhodno definirana barva `WEISS`.
- *Središče* navaja središče kroga.
- *Polmer* navaja polmer kroga.
- *Debelina* navaja debelino linije kroga. Če ta parameter izpustite ali pa ga nastavite na `0`, se krog napolni.

Ko je izpolnjen eden izmed pogojev `if`, so pike kocke najprej shranjene samo na virtualni risalni površini.

```
pygame.display.update()
```

Šele ta vrstica na koncu zanke posodobi grafiko na zaslonu. Nato lahko dejansko vidite pike kocke.



Slika 8.3: Šest možnih rezultatov metanja kocke.

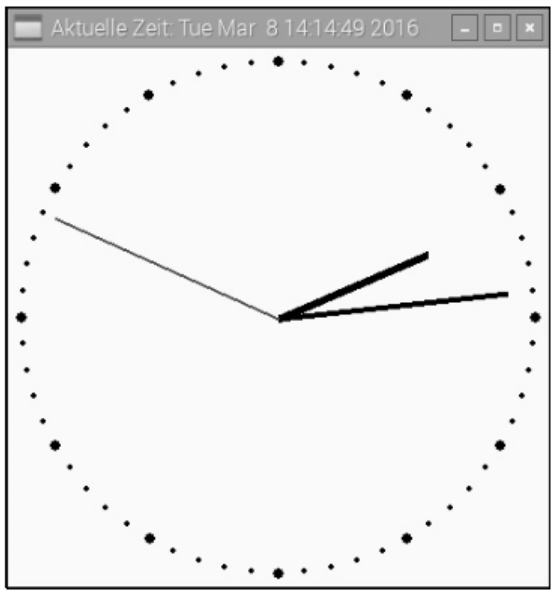
Zanka se takoj začne od začetka in ponovno čaka na pritisk tipke s strani uporabnika. Če se med izvajanjem zanke `mainloop` nastavi na `False`, ker želi uporabnik zaključiti z igro, se zanka več ne ponovi, temveč se izvede naslednja vrstica:

```
pygame.quit()
```

Ta vrstica zaključi modul PyGame, to pa zapre tudi grafično okno in nato še celoten program.

9 Analogna ura na zaslonu

Digitalni prikaz časa, ki smo ga danes vajeni pri računalnikih, je prišel v modo šele v 70. letih prejšnjega stoletja. Pred tem so čas več stoletij prikazovali analogno s kazalci na številčnici. Bum digitalnih ur se je v zadnjih letih ponovno malce zmanjšal, saj je bilo ugotovljeno, da je možno analogne ure hitreje odčitati, v slabih vremenskih pogojih ali na velikih razdaljah, kot je na primer na železniških postajah, pa je odčitavanje tudi bolj jasno. Človeško oko grafiko hitreje zajame kot številke ali črke. Slika analogne ure se vtisne v kratkoročni spomin, tako da je možno čas pravilno odčitati, tudi če smo sliko videli nepopolno ali megleno. Če pa nenatačno vidimo digitalno uro, iz tega ne moremo zanesljivo sklepati o prikazanem času.



Slika 9.1: Analogna ura, programirana s PyGame.

Ta program ni namenjen samo prikazu programiranja ure, temveč služi tudi ponazoritvi osnovnih načel za prikazovanje analognih prikazov, ki jih ni možno uporabljati samo za ure, temveč tudi za prikaz najrazličnejših izmerjenih vrednosti ali statističnih podatkov.

Okoli središča okrogle številčnice tečejo trije kazalci, ki prikazujejo ure, minute in sekunde. Zgoraj v naslovu okna teče tudi digitalni prikaz časa.

Program `uhr01.py` na zaslonu prikazuje analogno uro, ki jo vidite na sliki:

```
import pygame, time
from pygame.locals import *
from math import sin, cos, radians
pygame.init()
ROT = (255, 0, 0); WEISS = (255, 255, 255); SCHWARZ = (0, 0, 0)
FELD = pygame.display.set_mode((400, 400))
FELD.fill(WEISS)
MX = 200; MY = 200; MP = ((MX, MY))
def punkt(A, W):
     w1 = radians(W * 6 - 90); x1 = int(MX + A * cos(w1))
     y1 = int(MY + A * sin(w1)); return((x1, y1))
for i in range(60):
     pygame.draw.circle(FELD, SCHWARZ, punkt(190, i), 2)
for i in range(12):
     pygame.draw.circle(FELD, SCHWARZ, punkt(190, i * 5), 4)
```

```
mainloop = True; s1 = 0
while mainloop:
      zeit = time.localtime()
      s = zeit.tm_sec; m = zeit.tm_min; h = zeit.tm_hour
      if h > 12:
            h = h - 12
      hm = (h + m / 60.0) * 5
      if s1 <> s:
            pygame.draw.circle(FELD, WEISS, MP, 182)
            pygame.draw.line(FELD, SCHWARZ, MP, punkt(120, hm), 6)
            pygame.draw.line(FELD, SCHWARZ, MP, punkt(170, m), 4)
            pygame.draw.line(FELD, ROT, MP, punkt(180, s), 2)
            s1 = s
            pygame.display.set_caption("Aktuelle     Zeit:     "     +
            time.asctime())
            pygame.display.update()
            for event in pygame.event.get():
                  if event.type == QUIT or (event.type == KEYUP and
            event.key == K_ESCAPE):
                        mainloop = False
pygame.quit()
```

9.1.1 Tako deluje

Ta program prikazuje nadaljnje funkcije knjižnice PyGame in knjižnice `time` ter enostavne trigonometrične kotne funkcije, ki se uporabljajo za prikazovanje analognih prikazov.

```
import pygame, time
from pygame.locals import *
from math import sin, cos, radians
pygame.init()
```

Na začetku se tako kot v zadnjem programu inicializira knjižnica PyGame. Dodatno se uvozijo knjižnica `time` za določanje časa in tri funkcije iz zelo obsežne knjižnice `math`.

```
ROT = (255, 0, 0); WEISS = (255, 255, 255); SCHWARZ = (0, 0, 0)
```

Tri barve, ki se uporabijo v grafiki, se shranijo v treh spremenljivkah.

```
FELD = pygame.display.set_mode((400, 400)); FELD.fill(WEISS)
```

Odpre se okno, ki je veliko 400 x 400 slikovnih točk in je v celoti napolnjeno z belo barvo.

```
MX = 200; MY = 200; MP = ((MX, MY))
```

Tri spremenljivke določajo koordinate središča, po katerem se ravnajo vsi ostali grafični elementi, številčnica in kazalci. Spremenljivki `MX` in `MY` vsebujeta koordinati x in y središča, spremenljivka `MP` pa vsebuje središče kot točko, tako kot se uporablja za grafične funkcije.

Nato sledi definicija pomembne funkcije, ki na podlagi razdalje od središča in kota izračuna točke v koordinatnem sistemu. Ta funkcija se večkrat prikliče v programu za prikaz tako številčnice kot tudi kazalcev.

```
def punkt(A, W):
      w1 = radians(W * 6 - 90); x1 = int(MX + A * cos(w1))
      y1 = int(MY + A * sin(w1)); return((x1, y1))
```

Funkcija uporablja dva parametra: `A` je razdalja želene točke od središča, `W` pa je kot glede na središče. Za poenostavitev prikaza v primeru ure vzamemo kot v smeri urnega kazalca glede na navpično smer za 12:00. Kot se prav tako ne preda funkciji v stopinjah, temveč v minutah, in sicer 1/60 polnega kroga. Takšne predpostavke prihranijo številne vmesne izračune.

Tako kot večina programskih jezikov Python računa kotne enote v ločni meri in ne v stopinjah. Funkcija `radian()` iz knjižnice `math` preračuna stopinje v ločno mero. Pri tem se podatek o kotu v minutah, ki je uporabljen v programu, pomnoži s 6, da se dobi podatek o stopinjah, nato pa se 90 stopinj odšteje, tako da smer 0 kaže navpično navzgor, tako kot 0-ta minuta vsake ure. Ta kot, preračunan v ločno mero, se za nadaljnje izračune znotraj funkcije shrani v spremenljivko `w1`.

Prikaz analogne ure temelji na kotnih funkcijah sinus in kosinus. Z njuno pomočjo se iz kota točke v ločni meri glede na središče (`w1`) določijo njegove koordinate v pravokotnem koordinatnem sistemu (`x1` in `y1`). Koordinate središča se prevzamejo iz spremenljivk `MX` in `MY` , ki sta bili definirani izven funkcije in ki imata splošno veljavnost. Razdalja točke od središča se posreduje funkciji prek parametra `A`. Funkcija `int()` iz rezultata izračuna celoštevilsko vrednost, saj so lahko koordinate slikovnih točk podane samo kot celo število.

Povratna vrednost funkcije je geometrična točka z izračunanima koordinatama `x1` in `y1`, ki se tako kot vse točke nahaja v dvojnih narekovajih.

Po definiciji te funkcije se nariše številčnica.

```
for i in range(60):
      pygame.draw.circle(FELD, SCHWARZ, punkt(190, i), 2)
```

Zanka zaporedoma nariše `60` točk za prikaz minut na krogu. Vse točke se določijo s funkcijo `punkt()`. Imajo isto razdaljo od središča, ki je s `190` slikovnimi točkami v štirih kvadrantih še natančno 10 slikovnih točk oddaljena od roba okna. Točke imajo polmer `2` slikovni točki.

```
for i in range(12):
      pygame.draw.circle(FELD, SCHWARZ, punkt(190, i * 5), 4)
```

Druga zanka nariše `12` večjih krogov, ki označujejo ure na številčnici. Ti imajo polmer `4` slikovne točke in se preprosto narišejo čez obstoječe kroge ter jih popolnoma prekrijejo. Simboli za ure si sledijo v razdalji kotov po pet minutnih enot, kar je doseženo z množenjem s `5` v podatku o kotu, ki se posreduje funkciji.

```
mainloop = True; s1 = 0
```

Preden se začne glavna zanka programa, se definirata še dve pomožni spremenljivki, ki sta potrebni v nadaljnjem poteku. `mainloop` v zadnjem primeru programa navaja, če naj zanka deluje naprej ali želi uporabnik zaključiti program. `s1` shrani nazadnje prikazano sekundo.

```
while mainloop:
      zeit = time.localtime()
```

Sedaj se začne glavna zanka programa, ki v vsakem prehodu – ne glede na to, kako dolgo traja – zapiše trenutni čas v objekt `zeit`. Pri tem se uporabi funkcija `time.localtime()` iz knjižnice `time`. Rezultat je podatkovna struktura, ki je sestavljena iz različnih posameznih vrednosti.

```
s = zeit.tm_sec; m = zeit.tm_min; h = zeit.tm_hour
```

Tri vrednosti, ki so relevantne za kazalce – sekunde, minute in ure – se kot struktura zapišejo v tri spremenljivke `s, m` in `h`.

```
if h > 12:
     h = h - 12
```

Analogne ure prikazujejo samo dvanajst ur. Funkcija `time.localtime()` vse podatke o času nudi v 24-urnem formatu. Podatki o času popoldne se torej preprosto odštejejo za `12` ur.

Prikaz časa pri analognih urah
V odvisnosti od uporabljenega mehanizma pri analognih urah obstajata dva različna prikaza. Pri pravih analognih urah minutni kazalec izvaja enakomerno krožno premikanje, pri urah z digitalnim krmiljenjem, kot so na primer ure na železniških postajah, pa ob polni minuti skoči za celo minuto naprej. Drugi postopek ima to prednost, da je možno čas z enim pogledom odčitati do minute natančno. Delčki minute v vsakdanjiku ponavadi niso pomembne. Za naš program ure prav tako uporabljamo ta postopek. Ravno nasprotno pa urni kazalec mora izvajati enakomerno krožno premikanje. Tukaj bi bilo namreč zelo nenavadno in nepregledno, če bi kazalec vsako polno uro skočil naprej za celo uro.

```
hm = (h + m / 60.0) * 5
```
Spremenljivka `hm` shrani kot urnega kazalca v minutnih enotah, ki se uporabljajo v celotnem programu. Pri tem se k aktualnim uram prišteje 1/60 minutne vrednosti. V vsaki minuti se urni kazalec premakne naprej za 1/60 ure. Izračunana vrednost se pomnoži s `5`, saj se urni kazalec v eni uri pomakne naprej na številčnici za pet minutnih enot.

```
if s1 <> s:
```
Trajanje enega prehoda zanke v programu ni znano. Za analogno uro to pomeni, da se grafika ne posodablja pri vsakem prehodu zanke, temveč samo takrat, ko je aktualna sekunda drugačna od nazadnje narisane. Pri tem se kasneje v programu narisana sekunda shrani v spremenljivko `s1`, aktualna sekunda pa se vedno nahaja v spremenljivki `s`.

Ko se sekunda v primerjavi z nazadnje narisano sekundo spremeni, se grafika ure posodobi s pomočjo naslednjih navodil. Če se ne spremeni, potem posodabljanje grafike ni potrebno in zanka se začne od začetka z nadaljnjo poizvedbo glede aktualnega sistemskega časa.

```
pygame.draw.circle(FELD, WEISS, MP, 182)
```

Najprej se nariše bela površina kroga, ki popolnoma prekrije kazalce. Polmer je s 182 slikovnimi točkami malce večji od najdaljšega kazalca, da ga popolnoma prekrije. Risanje kroga čez celotno površino je bistveno enostavnejše od prebarvanja nazadnje narisanega kazalca do slikovne točke natančno.

```
pygame.draw.line(FELD, SCHWARZ, MP, punkt(120, hm), 6)
```

Ta vrstica nariše urni kazalec kot linijo širine 6 slikovnih točk z izhodiščem v središču in z dolžino `120` slikovnih točk. Risanje poteka v kotu, ki ga navaja spremenljivka `hm`. Funkcija `pygame.draw.line()` doslej ni bila uporabljena. Potrebuje pet parametrov:

• *Površina* nudi podatek o risalni površini, na kateri se riše, v našem primeru `FELD`.
• *Barva* navaja barvo kroga, v našem primeru je to predhodno definirana barva `SCHWARZ`.
• *Začetna točka* navaja začetno točko linije, v našem primeru je to središče ure.
• *Končna točka* navaja končno točko linije, v našem primeru se ta izračuna s funkcijo `punkt()` iz kota urnega kazalca.
• *Debelina* navaja debelino linije.

Ista funkcija nariše tudi ostala dva kazalca ure.

```
pygame.draw.line(FELD, SCHWARZ, MP, punkt(170, m), 4)
```

Ta vrstica nariše minutni kazalec kot linijo širine 4 slikovnih točk z izhodiščem v središču in z dolžino `170` slikovnih točk. Risanje poteka v kotu, ki ga navaja minutna vrednost.

```
pygame.draw.line(FELD, ROT, MP, punkt(180, s), 2)
```

Ta vrstica nariše sekundni kazalec kot rdečo linijo širine 2 slikovnih točk z izhodiščem v središču in z dolžino `180` slikovnih točk. Risanje poteka v kotu, ki ga navaja sekundna vrednost.

```
s1 = s
```
Sedaj se pravkar prikazana sekunda shrani v spremenljivki `s1`, da se lahko ta vrednost v naslednjih prehodih zanke primerja z aktualno sekundo.

```
pygame.display.set_caption("Aktuelle Zeit: " + time.asctime())
```

Ta vrstica zapiše trenutni čas v digitalni obliki v naslov okna. Pri tem se uporabi funkcija `time.asctime()` iz knjižnice `time`, ki nudi podatek o času kot že formatirano verigo znakov.

```
pygame.display.update()
```
Doslej so bili vsi grafični elementi narisani samo virtualno. Šele ta vrstica resnično na novo vzpostavi grafični prikaz. Posodobitev poteka hkrati. Iz tega razloga pri risanju posameznih kazalcev eden za drugim ne prihaja do migetanja slike.

```
for event in pygame.event.get():
     if event.type == QUIT or (event.type == KEYUP and event.key ==
K_ESCAPE):
          mainloop = False
```

Še vedno znotraj poizvedbe `if`, torej tudi samo enkrat na sekundo, sledi poizvedba o morebitnih sistemskih dogodkih, ki zahteva relativno veliko zmogljivost in s katero je možno določiti, če je želel uporabnik v roku zadnje sekunde zapreti okno z uro ali pa je pritisnil tipko `[Esc]`. Če se to zgodi, se spremenljivka `mainloop` nastavi na `False`, s čimer se zanka več ne aktivira ponovno.

```
pygame.quit()
```
Zadnja vrstica najprej zaključi modul PyGame, to pa zapre tudi grafično okno in nato še celoten program.

10 Grafična pogovorna polja za upravljanje programov

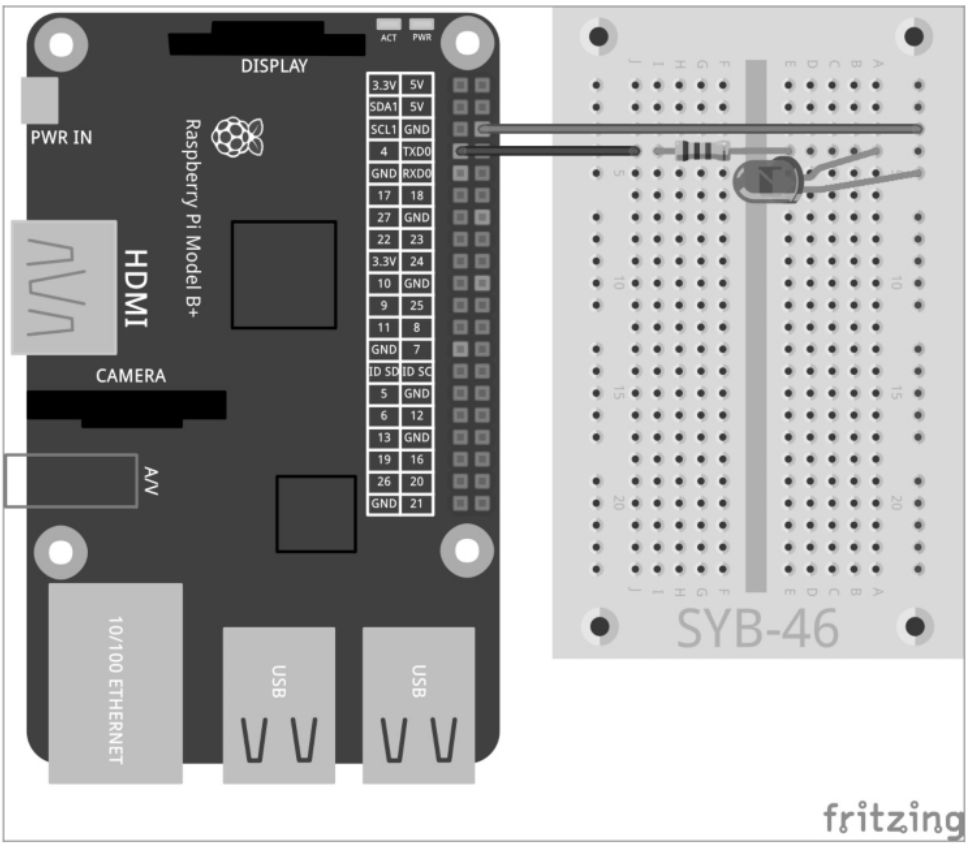
Noben sodobni program, ki zahteva kakršnokoli interakcijo z uporabnikom, ne deluje v čistem besedilnem načinu. Povsod so na voljo grafični vmesniki, na katerih lahko kliknete na gumbe in vam ni treba ničesar vnašati prek tipkovnice.

Python sam ne nudi grafičnih vmesnikov za programe, vendar obstaja več zunanjih modulov, ki so podobni že opisanemu modulu PyGame in so posebej namenjeni ustvarjanju grafičnih vmesnikov. Eden izmed najbolj znanih tovrstnih modulov je *Tkinter*, ki poskrbi, da je grafični vmesnik *Tk* – ki ga je možno uporabljati tudi za različne druge programske jezike – na voljo za Python.

Strukture grafičnega nabora orodij Tk se malce razlikujejo od Python in se na prvi pogled morda zdijo nenavadne. Iz tega razloga bomo začeli s popolnoma enostavnim primerom: Izklapljanje in vklapljanje LED prek gumbom v pogovornem polju.

Potrebne komponente:

1 x preizkusna ploščica
1 x rdeča LED
1 x 220 Ω upor
2 x priključni kabel



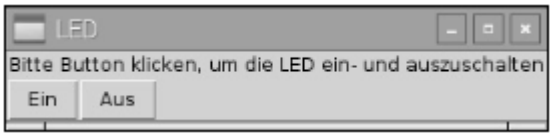
Slika 10.1: Ena sama LED na GPIO-vratih 4.

Priključite LED prek predupora na GPIO-vrata 4. Program `ledtk01.py` poskrbi, da sveti.

```
import RPi.GPIO as GPIO
from Tkinter import *
LED = 4; GPIO.setmode(GPIO.BCM); GPIO.setup(LED,GPIO.OUT)
def LedEin():
     GPIO.output(LED,True)

def LedAus():
     GPIO.output(LED,False)

root = Tk(); root.title("LED")
Label(root, text="Prosimo, da za vklop in izklop LED kliknete
gumb").pack()
Button(root, text="Ein", command=LedEin).pack(side=LEFT)
Button(root, text="Aus", command=LedAus).pack(side=LEFT)
root.mainloop()
GPIO.cleanup()
```



Slika 10.2: Tako bo videti gotovo pogovorno polje.

10.1.1 Tako deluje

Program prikazuje osnovne funkcije knjižnice Tkinter za izdelavo grafičnih pogovornih polj. Za razliko od grafične knjižnice PyGame, s katero poteka izdelava grafik do slikovne točke natančno, pa se velikost pogovornih polj in upravljalnih elementov v Tkinter samodejno določi glede na posamezno potrebno velikost, vendar pa jo lahko po potrebi tudi naknadno ročno spremenite.

```
import RPi.GPIO as GPIO
from Tkinter import *
```

Po uvozu GPIO-knjižnice se dodatno uvozijo še elementi knjižnice Tkinter.

```
LED = 4
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED,GPIO.OUT)
```

Te vrstice ne prikazujejo nič novega. GPIO-vrata `4` se definirajo kot izhodna vrata za LED in se označijo s spremenljivko `LED`.

```
def LedEin():
     GPIO.output(LED,True)
```

Sedaj se definira funkcija `LedEin(`, ki vklopi LED.

```
def LedAus():
     GPIO.output(LED,False)
```

Podobna funkcija, in sicer `LedAus()`, LED ponovno izklopi. Ti dve funkciji boste kasneje priklicali prek dveh gumbov v pogovornem polju.

Do sedaj je bil vse čisti Python, sedaj pa se nadaljuje s Tk in njegovimi posebnostmi.

```
root = Tk()
```
Tkinter uporablja tako imenovane gradnike. Pri tem gre za samostojne zaslonske elemente – v večini primerov so to pogovorna polja – ki vsebujejo različne elemente. Vsak program potrebuje gradnik `root`, iz katerega poteka priklic vseh ostalih objektov. Ta gradnik `root` se vedno imenuje `Tk()`, samodejno generira okno in tudi inicializira knjižnico Tkinter.

```
root.title("LED")
```
Objekti v Tkinter nudijo različne metode za različne namene. Metoda `title()` v gradniku nastavi naslov okna, v tem primeru torej zapiše besedo LED v naslovno vrstico novega okna.

Vsak gradnik lahko vsebuje več objektov, ki so posamezno definirani. Tkinter pri tem pozna različne tipe objektov, pri katerih vsak omogoča različne parametre, ki opisujejo lastnosti objekta. Parametri so navedeni v oklepaju za tipom objekta in so ločeni z vejicami. Ker lahko ta seznam postane zelo dolg, ponavadi vsak parameter zapišemo v lastno vrstico, tako da so vsi parametri poravnani eden pod drugim. Za razliko od zamikov pri zankah in poizvedbah v Python pa ti zamiki objektov iz knjižnice Tkinter niso obvezni.

```
Label(root, text="Prosimo, da za vklop in izklop LED kliknete
gumb").pack()
```

Objekti tipa `Label` so čista besedila v gradniku. Program jih lahko spreminja, vendar ne nudijo interakcije z uporabnikom. Prvi parameter v vsakem objektu Tkinter je ime nadrejenega gradnika, ponavadi okna, v katerem se nahaja posamezni objekt. V našem primeru je to edino okno v programu, in sicer gradnik `root`.

Parameter `text` vsebuje besedilo, ki naj bo prikazano na Label. Na koncu definicije objekta se doda tako imenovani packer kot metoda `pack()`. Ta packer vgradi objekt v pogovorno okno in generira geometrijo gradnika.

```
Button(root, text="Vklop", command=LedEin).pack(side=LEFT)
```

Objekti tipa `Button` so gumbi, na katere uporabnik klikne, da sproži določeno dejanje. Tudi tukaj parameter `text` vsebuje besedilo, ki naj bo prikazano na Button.

Parameter `command` vsebuje funkcijo, ki jo prikliče gumb, ko kliknete nanj. Pri tem ni možno posredovati nobenih parametrov, ime funkcije pa mora biti navedeno brez oklepajev. Ta gumb prikliče funkcijo `LedEin(`, ki vklopi LED.

Metoda `.pack()` lahko tudi vsebuje parametre, ki določajo, kako naj bo razporejen objekt znotraj pogovornega polja. `side=LEFT` pomeni, da bo gumb poravnan levo in ne na sredino.

```
Button(root, text="Izklop", command=LedAus).pack(side=LEFT)
```

Po isti shemi se ustvari še drugi gumb, ki prek funkcije `LedAus()` ponovno izklopi LED.

Sedaj so definirane vse funkcije in objekti, tako da se lahko začne dejanski program.

```
root.mainloop()
```
Glavni program je sestavljen samo iz ene vrstice. Začne glavno zanko `mainloop()`, in sicer gre za metodo gradnika `root`. Ta programska zanka čaka na to, da uporabnik aktivira enega izmed gradnikov in s tem sproži dejanje.

Simbola x zgoraj desno za zapiranje okna vam pri Tkinter ni treba posebej definirati. Ko uporabnik zapre glavno okno `root`, se glavna zanka `mainloop()` samodejno zaključi.

`GPIO.cleanup()`
Program teče dalje do zadnje vrstice in zapre odprta GPIO-vrata.

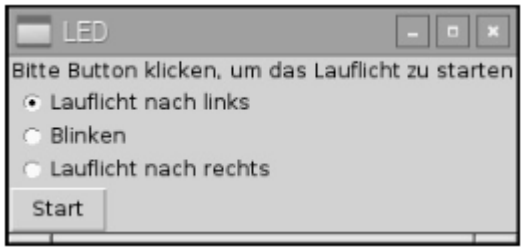
Po začetku programa se na zaslonu pojavi pogovorno polje. Za vklop LED kliknite na gumb *Vklop*, ko jo želite spet izklopiti, pa kliknite na gumb *Izklop*.

10.2 Upravljanje tekoče luči z grafičnim vmesnikom

Knjižnica Python Tkinter nudi še veliko več upravljalnih elementov kot samo enostavne gumbe. Prek radijskih gumbov lahko izdelujete izbirne menije, v katerih lahko uporabnik izbere eno izmed več ponujenih možnosti.

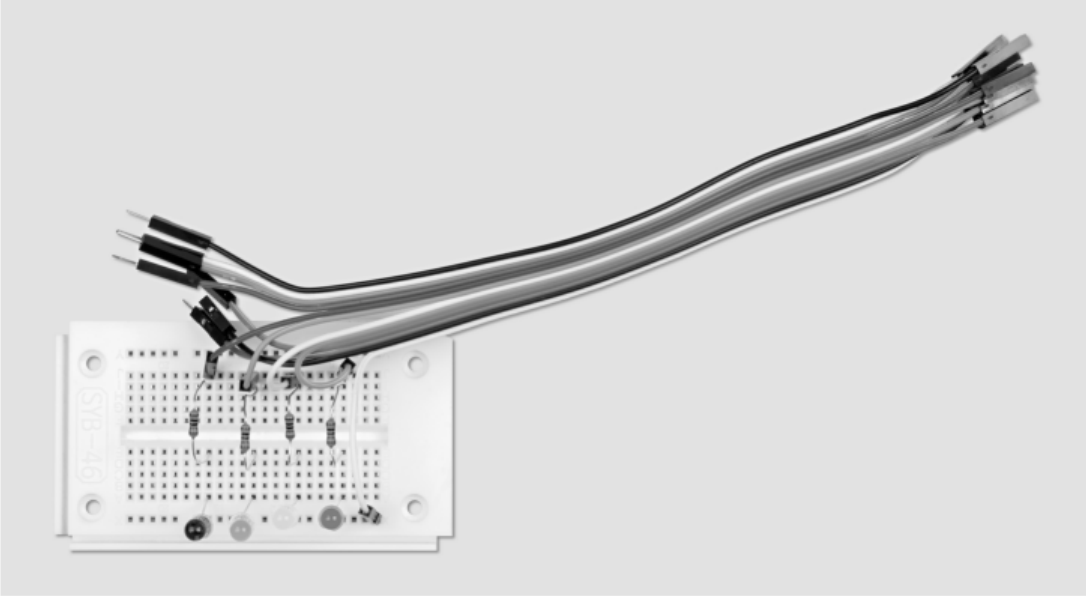
Kaj so radijski gumbi?
Ime "radijski gumb" dejansko izhaja iz starih radijskih sprejemnikov, ki so imeli tipke za izbiro predhodno nastavljenih radijskih postaj. Ob pritisku ene izmed teh tipk je tista tipka, ki je bila nazadnje pritisnjena, s pomočjo prefinjenega mehanizma samodejno spet izskočila. Radijski gumbi se odzivajo na enak način. Ko uporabnik izbere eno možnost, se druge samodejno deaktivirajo.

Naslednji preizkus prikazuje različne vzorce utripanja LED, ki so podobni tistim iz preizkusa "Pisani LED-vzorci in tekoče luči". Za razliko od tistih preizkusov uporabniku tukaj ni treba vnašati številk na besedilni zaslon, temveč lahko udobno izbere želeni vzorec iz enostavnega seznama.



Slika 10.3: Pogovorno polje omogoča izbiro med tremi LED-vzorci.

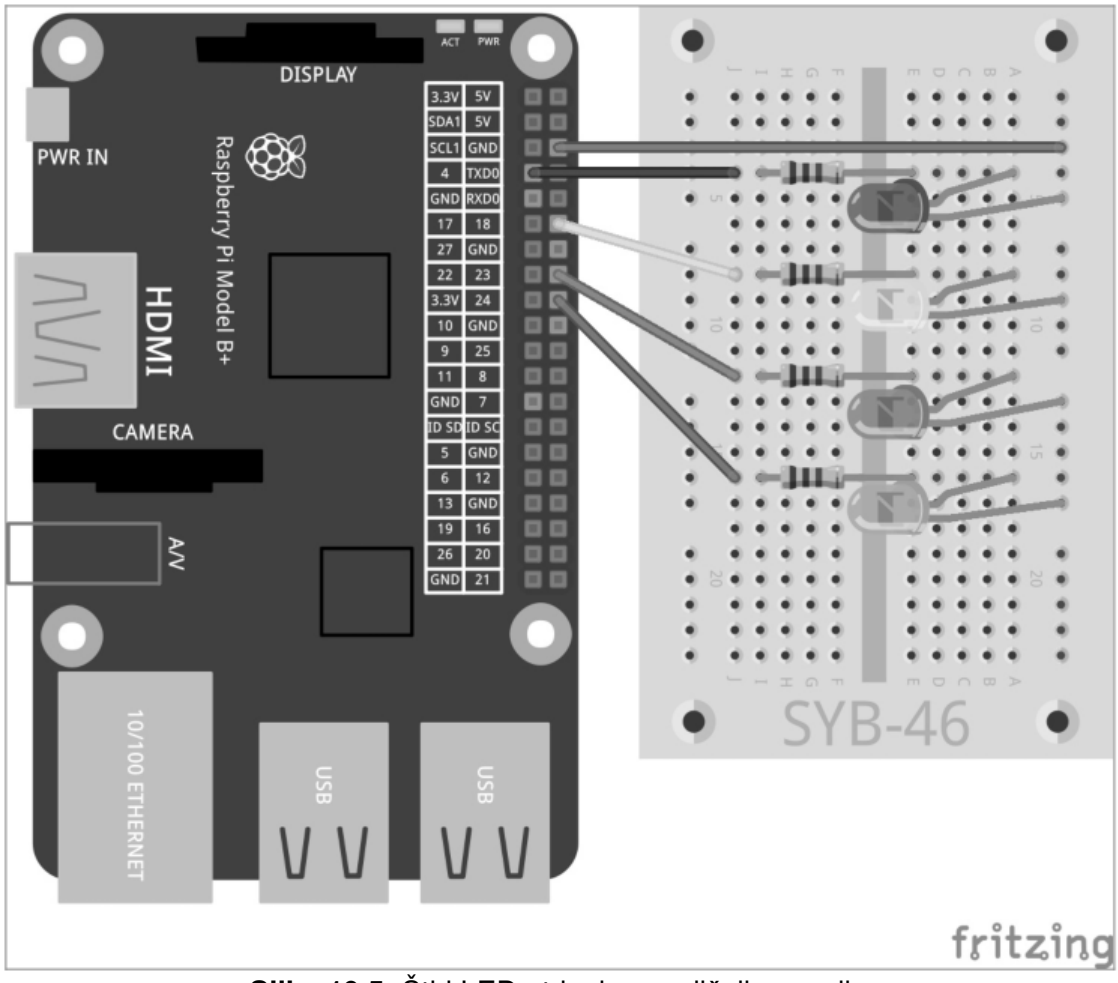
Sestav vezja je isti kot pri preizkusu "Pisani LED-vzorci in tekoče luči".



Slika 10.4: Sestav preizkusne ploščice pri preizkusu 10.2.

Potrebne komponente:

1 x preizkusna ploščica
1 x rdeča LED
1 x rumena LED
1 x zelena LED
1 x modra LED
4 x 220 Ω upor
5 x priključni kabel



Slika 10.5: Štiri LED utripajo v različnih vzorcih.

Program `ledtk02.py` temelji na prejšnjem programu, vendar je bil nadgrajen z radijskimi gumbi in funkcijami za LED tekoče luči in vzorce utripanja.

```
import RPi.GPIO as GPIO
import time
from Tkinter import *
GPIO.setmode(GPIO.BCM)
LED = [4,18,23,24]
for i in LED:
     GPIO.setup(i, GPIO.OUT, initial=0)
w = 5; t = 0.2
muster = [
     ("Tekoča luč v levo",1),
     ("Utripanje",2),
     ("Tekoča luč v desno",3)
          ]
root = Tk(); root.title("LED"); v = IntVar(); v.set(1)
def LedEin():
     e = v.get()
     if e == 1:
          for i in range(w):
               for j in range(4):
                    GPIO.output(LED[j], True)
                    time.sleep(t)
```

```
                    GPIO.output(LED[j], False)

    elif e == 2:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True)
        time.sleep(t)
        for j in range(4):
                GPIO.output(LED[j], False)
        time.sleep(t)
    else:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[3-j], True); time.sleep(t)
                GPIO.output(LED[3-j], False)
Label(root,
    text="Prosimo, da za aktivacijo tekoče luči kliknete
gumb").pack()
for txt, m in muster:
    Radiobutton(root, text = txt,
        variable = v, value = m).pack(anchor=W)
Button(root, text="Start", command=LedEin).pack(side=LEFT)
root.mainloop()
GPIO.cleanup()
```

10.2.1 Tako deluje

Na začetku se spet uvozijo vse potrebne knjižnice. Dodatno pri prejšnjem programu je tokrat zraven tudi knjižnica `time`, ki jo potrebujete za čakalne čase pri učinkih utripanja LED.

```
GPIO.setmode(GPIO.BCM); LED = [4,18,23,24]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)
```

Nato se definira seznam za štiri LED. Ustrezna GPIO-vrata se definirajo kot izhodi in nastavijo na 0, tako da so vse LED na začetku izklopljene.

```
w = 5; t = 0.2
```

Dve spremenljivki določata dve vrednosti v programu: število ponovitev `w` določenega vzorca in čas utripanja `t`. Obe vrednosti bi bilo možno tudi ob vsakem pojavu v programu fiksno vnesti. Vendar ju je možno na ta način enostavneje nastaviti, saj sta definirani samo na enem mestu.

```
muster = [
    ("Tekoča luč v levo",1), ("Utripanje",2), ("Tekoča luč v
    desno",3)
    ]
```

Besedila treh vzorcev, ki jih imate na izbiro, se definirajo v posebni obliki seznama. Vsak izmed treh elementov na seznamu je sestavljen iz para vrednosti. Posamezni par je sestavljen iz prikazanega besedila in številske vrednosti, ki se mora kasneje pri izbiri posameznega radijskega gumba vrniti nazaj.

```
root = Tk(); root.title("LED")
```

Inicializacija gradnika `root` se ponovno sklada s prejšnjim programom, razlika je samo v vsebinah pogovornega polja.

```
v = IntVar(); v.set(1)
```

Spremenljivke, ki se uporabljajo v pogovornih poljih Tk, je treba za razliko od običajnih spremenljivk Python pred prvo uporabo deklarirati. Ti dve vrstici deklarirata spremenljivko `v` kot celo število in jo na začetku nastavita na vrednost `1`.

```
def LedEin():
     e = v.get()
```

Sedaj se ponovno definira funkcija, ki se tako kot v prejšnjem primeru imenuje `LedEin()`, vendar pa se tokrat ne uporablja samo za vklop LED, temveč začne LED-vzorec. Druge funkcije `LedAus()` iz zadnjega primera tukaj ne potrebujemo. Prva vrstica nove funkcije prebere vnos uporabnika iz spremenljivke Tk `v` in zapiše vrednost v spremenljivko Python `e`. Kako vrednost prispe v spremenljivko `v`, boste izvedeli v nadaljevanju pri razlagi radijskega gumba.

V odvisnosti od izbire uporabnika se začnejo tri različne programske zanke:

```
if e == 1:
     for i in range(w):
          for j in range(4):
               GPIO.output(LED[j], True); time.sleep(t)
               GPIO.output(LED[j], False)
```

V prvem primeru se ena zanka petkrat ponovi in zaporedoma vklopi vsako izmed štirih LED, pri čemer posamezna LED 0,2 sekunde sveti in se nato ponovno izklopi. Pet ponovitev in čas utripanja 0,2 sekundi sta definirana s spremenljivkama `w` in `t` na začetku programa.

```
elif e == 2:
     for i in range(w):
          for j in range(4):
               GPIO.output(LED[j], True)
          time.sleep(t)
          for j in range(4):
               GPIO.output(LED[j], False)
          time.sleep(t)
```

V drugem primeru se petkrat zaporedoma hkrati vklopijo vse štiri LED. Potem ko 0,2 sekundi svetijo, se vse skupaj hkrati spet izklopijo.

```
else:
     for i in range(w):
          for j in range(4):
               GPIO.output(LED[3-j], True); time.sleep(t)
               GPIO.output(LED[3-j], False)
```

Tretji primer se sklada s prvim, vendar s to razliko, da se LED vklapljajo nazaj, s čimer tekoča luč teče v obratni smeri.

Potem ko je funkcija definirana, se ustvarijo elementi grafičnega vmesnika.

```
Label(root, text="Prosimo, da za aktivacijo tekoče luči kliknete gumb").pack()
```

Besedilo pogovornega polja se ponovno definira kot objekt `Label`. Nova je definicija treh radijskih gumbov.

```
for txt, m in muster:
     Radiobutton(root, text = txt, variable = v, value = m).pack(anchor=W)
```

Radijski gumbi se definirajo s posebno obliko zanke `for`. Namesto zančnega števca sta tukaj podani dve spremenljivki, ki se vzporedno štejeta. Oba števca zaporedoma stečeta skozi elemente seznama `muster`. Pri tem prva števna spremenljivka `txt` prevzame prvo vrednost para vrednosti: Pri tem gre za besedilo, ki je prikazano poleg radijskega gumba. Druga števna spremenljivka `m` prevzame številko posameznega vzorca iz druge vrednosti vsakega para vrednosti.

Zanka na ta način ustvari tri radijske gumbe, pri katerih je prvi parameter vedno `root` – gradnik, v katerem se nahajajo radijski gumbi. Parameter `text` radijskega gumba navaja besedilo, ki naj bo prikazano in ki se v našem primeru odčita iz spremenljivke `txt`. Parameter `variable` določi spremenljivko Tk, ki ste jo pred tem deklarirali in v katero se vnese vrednost radijskega gumba, ki ga je izbral uporabnik.

Parameter `value` za vsak radijski gumb določi številsko vrednost, ki se v našem primeru odčita iz spremenljivke `m`. Ko uporabnik klikne na ta radijski gumb, se vrednost parametra `value` zapiše v spremenljivko, ki je vnesena pod `variable`. Vsak izmed treh radijskih gumbov se v skladu s svojo definicijo vključi v pogovorno polje z metodo `pack()`. Parameter `anchor=W` poskrbi za to, da se radijski gumbi razporedijo eden pod drugim z levo poravnavo.

```
Button(root, text="Start", command=LedEin).pack(side=LEFT)
```

Gumb se definira tako kot v zadnjem primeru.

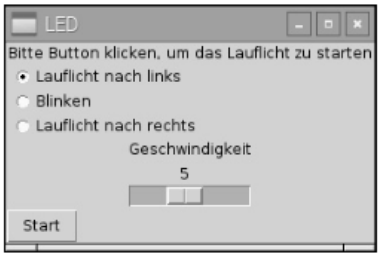
```
root.mainloop(); GPIO.cleanup()
```

Tudi glavna zanka in konec programa se skladata z zadnjim primerom.

Začnite program in prek enega izmed radijskih gumbov izberite želeni vzorec utripanja. Prek spremenljivke `v` je prva izbira že izbrana. Ko uporabljate radijske gumbe v pogovornem polju, vedno določite smiselno predhodno izbiro, tako da nikoli ne pride do nedefiniranega rezultata, če uporabnik sam ne izbere ničesar. Klik na *Start* nato aktivira izbrani vzorec in poskrbi za to, da petkrat steče. Nato lahko izberete drugi vzorec.

10.3 Nastavitev hitrosti utripanja

V tretjem koraku se pogovorno polje ponovno nadgradi. Uporabnik lahko sedaj nastavi hitrost utripanja z drsnim regulatorjem.



Slika 10.6: Izbira med tremi LED-vzorci in nastavljiva hitrost utripanja.

Uporaba drsnih regulatorjev
Drsni regulatorji nudijo zelo intuitivno metodo za vnos številskih vrednosti znotraj določenega območja. Na ta način si prihranite verjetnostno poizvedbo, ki ugotovi, če je uporabnik vnesel vrednost, ki jo lahko program tudi smiselno udejanji, saj vrednosti izven območja, ki ga določa drsni regulator, niso možne. Drsni regulator vedno nastavite tako, da si lahko uporabnik predstavlja vrednosti. Nima smisla, da omogočite nastavljanje vrednosti na območju milijonov. Če sama absolutna številska vrednost nima dejanske vloge, uporabniku enostavno ponudite lestvico od 1 do 10 ali 100 in ustrezno preračunajte vrednost v programu. Vrednosti morajo naraščati od leve proti desni, saj je obratni vrstni red za večino uporabnikov nenavaden. Poleg tega vedno določite smiselno vrednost, ki se nastavi, če uporabnik ne premika drsnega regulatorja.

Program `ledtk03.py` se v veliki meri sklada s prejšnjim primerom, doda se samo regulacija hitrosti.

```
import RPi.GPIO as GPIO
import time
from Tkinter import *

GPIO.setmode(GPIO.BCM); LED = [4,18,23,24]
for i in LED:
     GPIO.setup(i, GPIO.OUT, initial=0)

w = 5
muster = [
     ("Tekoča luč v levo",1), ("Utripanje",2), ("Tekoča luč v
     desno",3)
     ]

root = Tk(); root.title("LED"); v = IntVar(); v.set(1); g =
IntVar(); g.set(5)

def LedEin():
     e = v.get()
     t = 1.0/g.get()
     if e == 1:
          for i in range(w):
               for j in range(4):
                    GPIO.output(LED[j], True); time.sleep(t)
                    GPIO.output(LED[j], False)
     elif e == 2:
          for i in range(w):
               for j in range(4):
                    GPIO.output(LED[j], True)
               time.sleep(t)
```

```
                for j in range(4):
                    GPIO.output(LED[j], False)
                time.sleep(t)
    else:
        for i in range(w):
                for j in range(4):
                    GPIO.output(LED[3-j], True): time.sleep(t)
                    GPIO.output(LED[3-j], False)

Label(root,text="Prosimo, da za aktivacijo tekoče luči kliknete
gumb").pack()

for txt, m in muster:
    Radiobutton(root, text = txt, variable = v, value =
    m).pack(anchor=W)

Label(root, text="Geschwindigkeit").pack()

Scale(root, orient=HORIZONTAL, from_ = 1, to = 10, variable =
g).pack()

Button(root, text="Start", command=LedEin).pack(side=LEFT)

root.mainloop()
GPIO.cleanup()
```

10.3.1 Tako deluje

Inicializacija knjižnic in GPIO-vrat ter definicija seznama za tri vzorce utripanja se skladajo s prejšnjim programom. Določitev spremenljivke `t` za čas utripanja odpade, saj se ta kasneje odčita iz drsnega regulatorja.

```
g = IntVar(); g.set(5)
```

Dodatno k spremenljivki Tk `v`, v katero se shrani izbrani vzorec utripanja, se deklarira dodatna spremenljivka s celoštevilsko vrednostjo `g` za hitrost. Ta vsebuje začetno vrednost `5`, ki se sklada s srednjo vrednostjo drsnega regulatorja.

```
def LedEin():
    e = v.get(); t = 1.0/g.get()
```

Funkcija, ki poskrbi za utripanje LED, se prav tako sklada s prejšnjim primerom, vendar z eno razliko. Spremenljivka `t` za čas utripanja se razbere iz vrednosti drsnega regulatorja `g`.

Ker uporabnik intuitivno poveže hitrejše utripanje z višjo hitrostjo, bo drsni regulator s pomikanjem v desno nudil višje vrednosti. Vendar pa je za večjo hitrost v programu treba nastaviti krajši čakalni čas, torej nižjo vrednost. To dosežete z izračunom obratne vrednosti, ki na osnovi vrednosti od `1` do `10` drsnega regulatorja določi vrednosti od `1.0` do `0.1` za spremenljivko `t`. V formuli se mora nahajati `1.0` in ne `1`, tako da je rezultat število s plavajočo vejico in ne celo število.

Preračunavanje celih števil v števila s plavajočo vejico
Rezultat izračuna se samodejno shrani kot število s plavajočo vejico, ko je najmanj ena izmed vrednosti v formuli število s plavajočo vejico. Ko so vse vrednosti v formuli cela števila, se rezultat prav tako skrajša na celo število.

Definicija oznake in radijskih gumbov v pogovornem polju se prevzame iz prejšnjega primera.

```
Label(root,
      text="Hitrost").pack()
```

Za razlago drsnega regulatorja se v pogovorno polje zapiše dodatna oznaka. Ker ne vsebuje nobenih parametrov v metodi `pack()`, se vstavi vodoravno centrirano pod radijskimi gumbi.

```
Scale(root, orient=HORIZONTAL, from_ = 1, to = 10, variable = g).pack()
```

Drsni regulator je objekt tipa `Scale`, ki tako kot vsi objekti v tem pogovornem polju kot prvi parameter vsebuje parameter `root`. Parameter `orient=HORIZONTAL` nudi podatek o tem, da se drsni regulator nahaja v vodoravnem položaju. Brez tega parametra bi se nahajal v navpičnem položaju. Parameter `from_` in `to` navajata začetno in končno vrednost drsnega regulatorja. Pri tem upoštevajte način pisanja `from_`, saj je `from` brez podčrtaja v Python rezervirana vrednost za uvoz knjižnic. Parameter `variable` določi predhodno deklarirano spremenljivko Tk, v katero se vnese trenutno nastavljena vrednost drsnega regulatorja. Začetna vrednost se prevzame iz vrednosti, ki se določi pri deklaraciji spremenljivke, v tem primeru je to `5`.

Drsni regulator se z metodo `pack()` ponovno vstavi vodoravno centrirano v pogovorno polje.

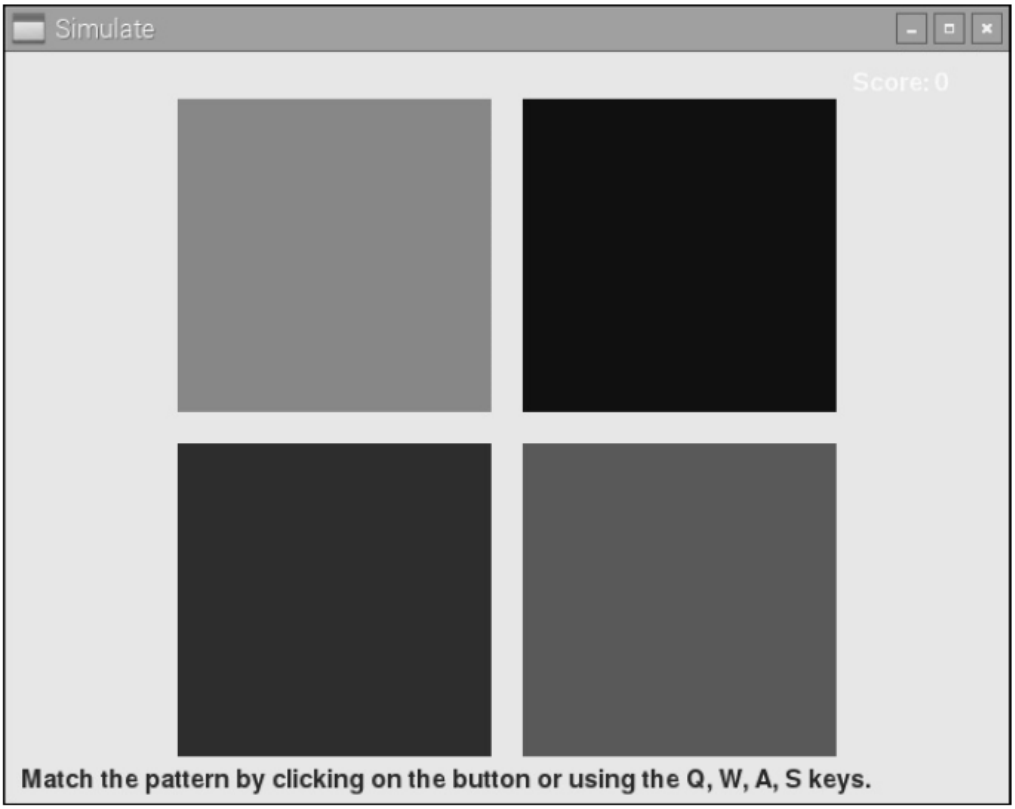
Ostali deli programa – gumb *Start*, glavna zanka in konec programa – se nespremenjeni prevzamejo iz prejšnjega primera.

Začnite program, izberite vzorec utripanja in določite hitrost. Višje vrednosti poskrbijo za hitrejše utripanje vzorcev. Pri kliku na gumb *Start* funkcija `LedEin()` odčita izbrani vzorec utripanja iz radijskih gumbov in hitrost iz položaja drsnega regulatorja.

11 PiDance z LED

V poznih 70. letih prejšnjega stoletja, še pred časom pravih računalniških iger, je obstajala elektronska igra s štirimi barvnimi lučkami, ki je bila leta 1979 v prvem tovrstnem izboru imenovana za najboljšo igro leta. Igra se je v Nemčiji tržila pod imenom *Senso*. Atari je izdal repliko z imenom *Touch Me* v velikosti žepnega kalkulatorja. Pojavila se je še ena replika z imenom *Einstein*, na angleško govorečem tržišču pa se je igra Senso tržila kot *Simon*.

Raspbian nudi grafično različico te igre pri *Python Games* pod imenom *Simulate*.

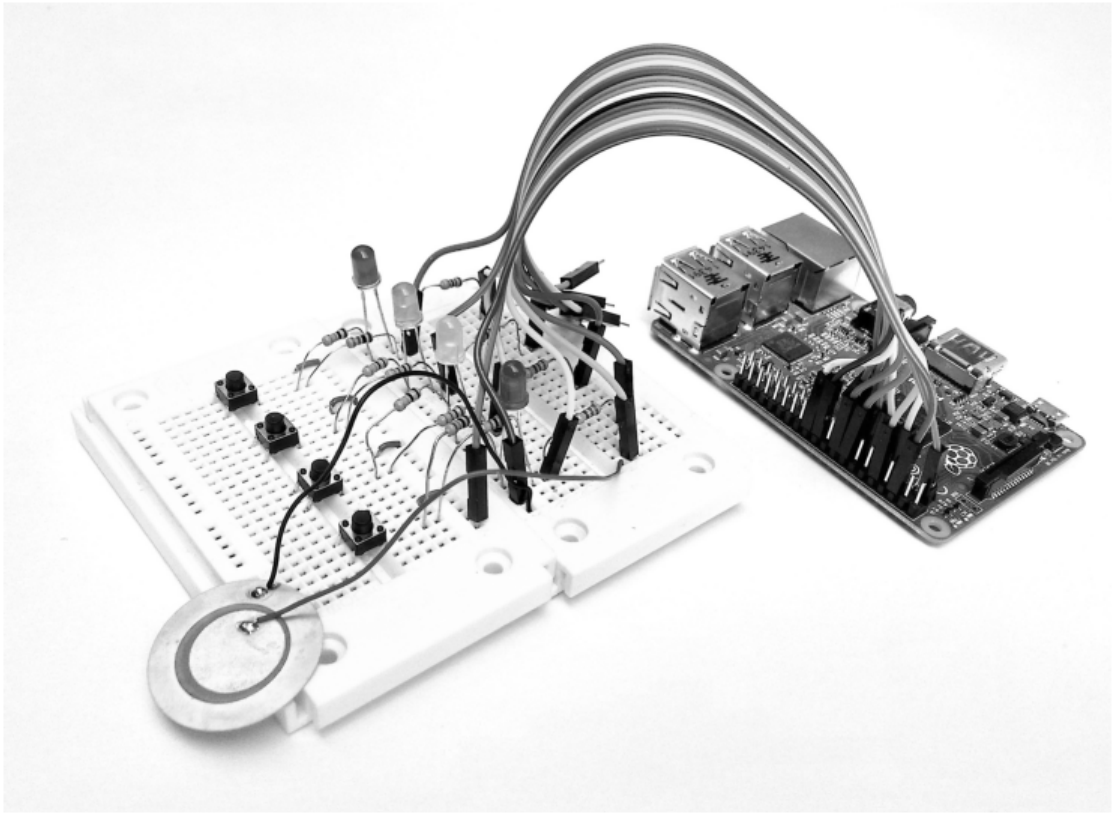


Slika 11.1: Igra Simulate iz Python Games.

Naša igra PiDance prav tako temelji na principu te igre. LED utripajo v naključnem vrstnem redu. Uporabnik mora nato pritisniti enako zaporedje s pomočjo tipk. V vsakem naslednjem krogu sveti dodatna LED, tako da si vedno težje zapomnite zaporedje. Takoj ko naredite napako, je igre konec.

Igra se sestavi na dveh preizkusnih ploščicah, tako da se tipkala nahajajo na robu in jih je možno dobro upravljati, ne da bi pri tem pomotoma izvlekli kable iz preizkusnih ploščic. Za boljšo stabilnost lahko preizkusni ploščici na vzdolžnih straneh nataknete eno na drugo.

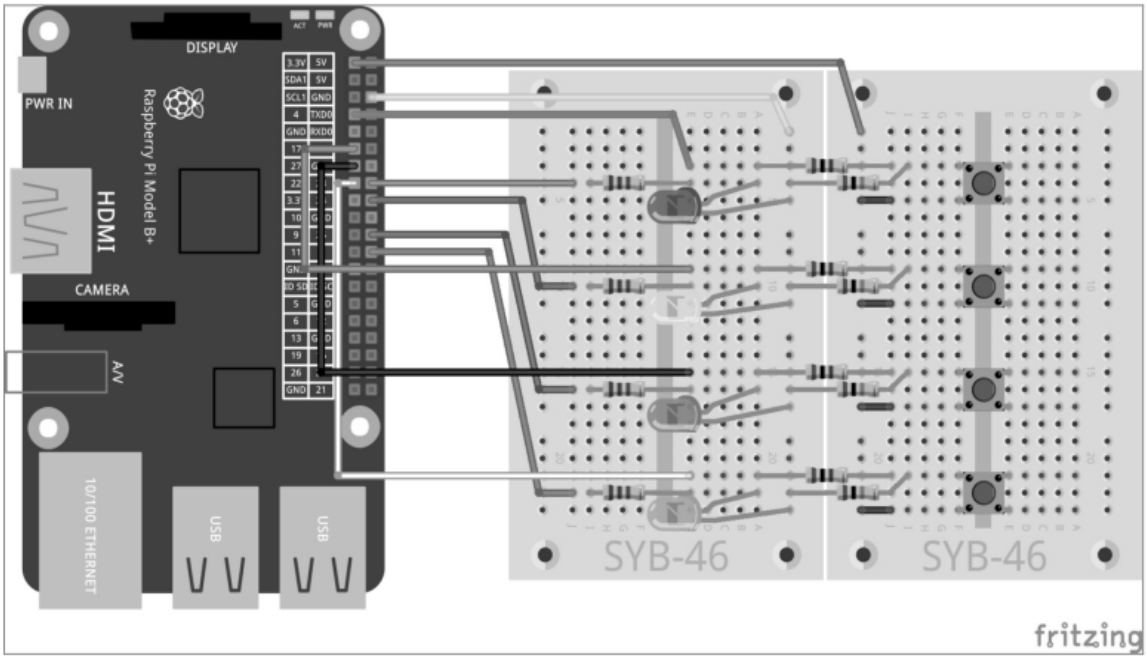
Dodatno poleg že znanih priključnih kablov potrebujete še štiri kratke žične mostičke. Pri tem z ostrimi kleščami ali škarjami za žico narežite priloženo stikalno žico na pribl. 2,5 cm dolge kose in s pomočjo ostrega rezila odstranite izolacijo na obeh koncih na dolžini pribl. 7 mm. Te kose žice upognite v obliki črke U. Nato lahko z njimi povežete po dve vrsti na preizkusni ploščici.



Slika 11.2: Sestav preizkusne ploščice pri preizkusu št. 11.

Potrebne komponente:

2 x preizkusna ploščica
1 x rdeča LED
1 x rumena LED
1 x zelena LED
1 x modra LED
4 x 220 Ω upor
4 x 1 kΩ upor
4 x 10 kΩ upor
4 x tipkalo
10 x priključni kabel
4 x kratek žični mostiček



Slika 11.3: PiDance z LED in tipkali na dveh preizkusnih ploščicah.

Tipkala so pritrjena nasproti ustreznih LED. Dve sredinski vzdolžni vrsti preizkusnih ploščic na obeh straneh mesta povezave služita kot 0 V in +3,3 V kabel za vezje.

Program `pidance01.py` vsebuje gotovo igro.

```
# -*- coding: utf-8 -*-
import time, random
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
rzahl = 10; farbe = []
for i in range(rzahl):
      farbe.append(random.randrange(4))
LED = [23,24,25,8]
for i in LED:
      GPIO.setup(i, GPIO.OUT, initial=False)
TAST = [4,17,21,22]
for i in TAST:
      GPIO.setup(i, GPIO.IN)
def LEDein(n, z):
      GPIO.output(LED[n], True); time.sleep(z)
      GPIO.output(LED[n], False); time.sleep(0.15)
def Druecken():
      while True:
            if(GPIO.input(TAST[0])):
                  return 0
            if(GPIO.input(TAST[1])):
                  return 1
            if(GPIO.input(TAST[2])):
                  return 2
            if(GPIO.input(TAST[3])):
                  return 3
ok = True
```

```
for runde in range(1, rzahl +1):
      print "Krog", runde
      for i in range(runde):
            LEDein(farbe[i], 1)
      for i in range(runde):
            taste = Druecken()
            LEDein(taste, 0.2)
            if(taste != farbe[i]):
                  print "Izgubili ste!"
                  print "Dosegli ste krog ", krog - 1, " "
                  for j in range(4):
                        GPIO.output(LED[j], True)
                  for j in range(4):
                        time.sleep(0.5)
                        GPIO.output(LED[j], False)
                  ok = False
                  break
      if(ok == False):
            break
      time.sleep(0.5)
if(ok == True):
      print "Odlično opravljeno!"
      for i in range(5):
            for j in range(4):
                  GPIO.output(LED[j], True)
            time.sleep(0.05)
            for j in range(4):
                  GPIO.output(LED[j], False)
            time.sleep(0.05)
GPIO.cleanup()
```

11.1.1 Tako deluje

Program nudi veliko novega, vendar so osnove GPIO-krmiljenja znane.

```
rzahl = 10
```

Po uvozu modulov `time`, `random` in `RPi.GPIO` se ustvari spremenljivka `rzahl`, ki določi število razpoložljivih krogov igre. Seveda lahko igrate tudi več kot deset krogov – več krogov kot odigrate, težje si je zapolniti zaporedje utripanja.

```
farbe = []
for i in range(rzahl):
      farbe.append(random.randrange(4))
```

Seznam `farbe` se prek zanke napolni s tolikšnim številom naključnih števil med 0 in 3, kolikor krogov igrate. Pri tem se uporabi metoda `append()`, ki je na voljo v vsakem seznamu. Ta na seznam doda element, ki je bil posredovan kot parameter.

```
LED = [23,24,25,8]
for i in LED:
      GPIO.setup(i, GPIO.OUT, initial=False)
```

GPIO-vrata za LED se po znani shemi nastavijo kot izhodi v seznamu `LED` in so vsi izklopljeni.

```
TAST = [4,17,21,22]
for i in TAST:
     GPIO.setup(i, GPIO.IN)
```

Po istem principu se GPIO-vrata za štiri tipkala nastavijo kot vhodi v seznamu `TAST`.

S tem je poskrbljeno za osnove, poleg tega pa se definirata še dve funkciji, ki ju boste večkrat potrebovali v programu.

```
def LEDein(n, z):
     GPIO.output(LED[n], True); time.sleep(z)
     GPIO.output(LED[n], False); time.sleep(0.15)
```

Funkcija `LEDein()` vključi LED in poskrbi, da nekaj časa sveti. Funkcija uporablja dva parametra. Prvi parameter, `n` nudi podatek o številu LED med 0 in 3, drugi parameter `z` pa navaja čas, kako dolgo naj LED sveti. Ko se LED spet izklopi, funkcija še 0,15 sekunde počaka in se šele nato zaključi, tako da pri večkratnem priklicu vidite kratke premore med svetenjem posameznih LED. To je posebej pomembno, ko ena LED večkrat zaporedoma zasveti. To sicer ne bi bilo vidno.

```
def Druecken():
     while True:
          if(GPIO.input(TAST[0])):
               return 0
          if(GPIO.input(TAST[1])):
               return 1
          if(GPIO.input(TAST[2])):
               return 2
          if(GPIO.input(TAST[3])):
               return 3
```

Funkcija `Druecken()` je sestavljena iz neskončne zanke, ki čaka na to, da uporabnik pritisne enega izmed tipkal. Nato se številka tipkala posreduje glavnemu programu.

```
ok = True
```

Po definiciji funkcij se začne dejanski glavni program in najprej nastavi spremenljivko `ok` na `True`. Takoj ko igralec naredi napako, se `ok` nastavi na `False`. Če je spremenljivka po nastavljenem številu krogov še vedno nastavljena na `True`, potem je igralec zmagal.

```
for runde in range(1, rzahl +1):
```

Igra ima toliko krogov, kolikor jih je nastavljenih v spremenljivki `rzahl`. Števec krogov se pri tem prestavi za 1 navzgor, tako da se igra začne s krogom št. 1 in ne s krogom št. 0.

```
print "Krog", runde
```

Trenutni krog je prikazan v oknu Python Shell.

```
for i in range(runde):
     LEDein(farbe[i], 1)
```

Sedaj program predvaja vzorec, ki si ga mora igralec zapomniti. V odvisnosti od trenutnega števila krogov zaporedoma zasveti ustrezno število LED z naključno izbranimi barvami v skladu s seznamom `farbe`, ki je bil določen na začetku programa. Ker se števec `runde`

začne z 1, že v prvem krogu sveti točno 1 LED. Za aktivacijo svetenja LED se uporabi funkcija `LEDein()`, katere prvi parameter je barva iz ustreznega položaja na seznamu, drugi parameter pa poskrbi, da vsaka LED sveti eno sekundo.

```
for i in range(runde):
```
Po predvajanju barvnega vzorca se začne dodatna zanka, v kateri mora igralec prek tipkal ponovno vnesti isti vzorec iz spomina.

```
taste = Druecken()
```
Pri tem se prikliče funkcija `Druecken()`, ki čaka, dokler igralec ne pritisne tipkala. Številka pritisnjenega tipkala se shrani v spremenljivko `taste`.

```
LEDein(taste, 0.2)
```
Po pritisku tipke ustrezna LED kratko zasveti za 0,2 sekundi.

```
if(taste != farbe[i]):
```
Če se nazadnje pritisnjena tipka ne sklada z barvo na ustreznem položaju na seznamu, je igralec izgubil. Operator `!=` pomeni neenako. Tukaj lahko uporabite tudi <>.

```
print "Izgubili ste!"
print "Dosegli ste krog ", krog - 1, " "
```

Program na zaslonu prikaže, da je igralec izgubil in koliko krogov je uspešno opravil. Število opravljenih krogov je za ena nižje od trenutnega števca krogov.

```
for j in range(4):
     GPIO.output(LED[j], True)
```

Kot optično vidno znamenje se vklopijo vse LED ...

```
for j in range(4):
     time.sleep(0.5); GPIO.output(LED[j], False)
```

... nato pa se na za drugo spet izklopijo v zamiku 0,5 sekund. Rezultat je jasen učinek zniževanja.

```
ok = False
```
Spremenljivka `ok`, ki označuje, če je igralec še v igri, se prestavi na `False` ...

```
break
```
... in prekine zanko. Igralec ne more več pritisniti nobene tipke. Pri prvi napaki je takoj konec.

```
if(ok == False):
     break
```

Ko se `ok` nahaja na `False`, potem se prekine tudi zunanja zanka in ne sledi več noben krog.

```
time.sleep(0.5)
```
Ko je bil vnos zaporedja pravilen, program 0,5 sekunde počaka, nato pa se začne naslednji krog.

```
if(ok == True):
```
Program doseže to točko, če je torej zanka v celoti stekla, torej če je igralec pravilno vnesel vsa zaporedja, ali pa je bila prejšnja zanka prekinjena zaradi napake, ki jo je naredil igralec.

Če se `ok` še vedno nahaja na `True`, sledi potrditev zmagovalca. V nasprotnem primeru se ta blok preskoči in igra izvede samo še zadnjo programsko vrstico.

```
print "Odlično opravljeno!"
for i in range(5):
      for j in range(4):
            GPIO.output(LED[j], True)
      time.sleep(0.05)
      for j in range(4):
            GPIO.output(LED[j], False)
      time.sleep(0.05)
```

Če igralec zmaga, se v oknu Python Shell pojavi ustrezno sporočilo. Nato vse LED petkrat zaporedoma kratko utripajo.

```
GPIO.cleanup()
```
Zadnja vrstica se vedno izvede. Pri tem se zaprejo uporabljena GPIO-vrata.

Kolofon

www.elo-web.de
Avtor: Christian Immler

ISBN 978-3-645-10145-5

Odstranjevanje

Električne in elektronske naprave ni dovoljeno metati med gospodinjske odpadke!

Izdelek po izteku njegove življenjske dobe odstranite v skladu z veljavnimi zakonskimi predpisi. Obstajajo lokalna zbirališča odpadkov, na katerih lahko brezplačno oddate odslužene električne naprave. O lokaciji tovrstnih zbirališč se pozanimajte na svoji občini.

Izjava o skladnosti

Ta izdelek je v skladu z veljavnimi direktivami CE, v kolikor ga uporabljate v skladu s priloženimi navodili za uporabo. Ta navodila za uporabo sodijo k izdelku. Če izdelek predate v uporabo tretji osebi, priložite tudi ta navodila.

Pozor! Zaščita oči in LED

Ne glejte neposredno v LED z majhne razdalje, saj lahko z neposrednim gledanjem pride do poškodb mrežnice! To velja predvsem za svetle LED v prozornem ohišju ter v posebni meri za zmogljive LED. Pri belih, modrih, vijoličnih in ultravijoličnih LED daje navidezna svetilnost napačen vtis o dejanski nevarnosti za vaše oči. Posebna previdnost je potrebna pri uporabi zbiralnih leč. LED uporabljajte tako kot je opisano v teh navodilih za uporabo. Za napajanje ne uporabljajte večjih tokov od priporočenih.

Ta navodila za uporabo so publikacija podjetja Conrad Electronic d.o.o. k.d., Ljubljanska cesta 66, 1290 Grosuplje.

GARANCIJSKI LIST

Conrad Electronic d.o.o. k.d.
Ljubljanska c. 66, 1290 Grosuplje
Fax: 01/78 11 250, Tel: 01/78 11 248
www.conrad.si, info@conrad.si

Izdelek: **Učni komplet za Raspberry Pi Conrad Components**
Kat. št.: **12 25 953**

Garancijska izjava:
Proizvajalec jamči za kakovost oziroma brezhibno delovanje v garancijskem roku, ki začne teči z izročitvijo blaga potrošniku. **Garancija velja na območju Republike Slovenije. Garancija za izdelek je 1 leto**.
Izdelek, ki bo poslan v reklamacijo, vam bomo najkasneje v skupnem roku 45 dni vrnili popravljenega ali ga zamenjali z enakim novim in brezhibnim izdelkom. Okvare zaradi neupoštevanja priloženih navodil, nepravilne uporabe, malomarnega ravnanja z izdelkom in mehanske poškodbe so izvzete iz garancijskih pogojev. **Garancija ne izključuje pravic potrošnika, ki izhajajo iz odgovornosti prodajalca za napake na blagu**.
Vzdrževanje, nadomestne dele in priklopne aparate proizvajalec zagotavlja še 3 leta po preteku garancije.
Servisiranje izvaja proizvajalec sam na sedežu firme CONRAD ELECTRONIC SE, Klaus-Conrad-Strasse 1, Nemčija.
Pokvarjen izdelek pošljete na naslov: Conrad Electronic d.o.o. k.d., Ljubljanska cesta 66, 1290 Grosuplje, skupaj z izpolnjenim garancijskim listom.

Prodajalec: _____

Datum izročitve blaga in žig prodajalca:
_____

Garancija velja od dneva izročitve izdelka, kar kupec dokaže s priloženim, pravilno izpolnjenim garancijskim listom.