Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Darja Peternel

Raspberry Pi kot okolje za učenje programiranja

diplomsko delo

interdisciplinarni univerzitetni študijski program prve stopnje računalništvo in matematika

Mentor: izr. prof. dr. Janez Demšar

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Učenje programiranja je zahtevno, zato je potrebno učence zanj motivirati s primernim okoljem. V drugem triletju osnovne šole lahko v ta namen uporabljamo Scratch, za starejše učence pa le-ta ni več zanimiv. Ena od aktualnih možnosti je uporaba mikrokontrolerjev ali mini računalnikov, na primer Arduino in Raspberry, na katere lahko učenci priključujejo različne naprave, kot so diode, releji, motorji ter različni senzorji in tipke.

Pripravite zbirko nalog iz programiranja Raspberry Pi. Naloge naj imajo rdečo nit, primerno za učence v tretjem triletju osnovne šole.

Zahvaljujem se mentorju, izr. prof. dr. Janezu Demšarju, za strokovno vodenje in pomoč pri izdelavi diplomske naloge.

Posebna zahvala gre tudi moji družini ter najbližjim prijateljem, ki so me podpirali ter mi stali ob strani skozi vsa leta študija.

Kazalo

Povzetek

Diplomska naloga se ukvarja s projekti, ki bi otroke zadnje triade osnovne šole spodbudili k programiranju, ko programski jezik Scratch postane preveč omejujoč. Zadnje čase se v odgovor ponuja uporaba pravih programskih jezikov in mikrokontrolerjev, na katere lahko priključimo kamero, senzorje in ostale naprave. S tem otroke naučimo osnov elektrotehnike, od koder manjka le korak do izdelave lastne igre oziroma manjšega projekta. V diplomski nalogi so predstavljene sestavljene naloge, ki otroke in ostale začetnike s pomočjo mikrokontrolerja Raspberry Pi in igre Minecraft na zabaven način vpeljejo v programiranje s programskim jezikom Python. Ta predstavlja pravšnjo mero izziva in je zaradi enostavne strukture in jasno definirane sintakse primeren za poučevanje programiranja.

Ključne besede: učenje programiranja, Raspberry Pi, Python.

Abstract

The thesis discusses the projects which might encourage primary school children in the third triad to programme when the programming language Scratch becomes too restrictive. Lately, the answer has been to use proper programming languages and microcontrollers, which can be connected to camera, sensors and other devices. With this, children learn the basics of electrical engineering, and are only one step away from creating their own game or a small project. The thesis consists of assignments, which with the microcontroller Raspberry Pi and the game Minecraft in a fun way lead children and other beginners to programme with Python. The language is suitable for learning programming as it offers the right amount of challenge, and has a simple structure and a clear defined syntax.

Keywords: learning programming, Raspberry Pi, Python.

Poglavje 1

Uvod

Programiranje je dajanje navodil računalniku v jeziku, ki ga ta razume. Otroci se hitro in brez eksplicitnega učenja naučijo jezika, saj se nahajajo v okolju, v katerem je uporaba jezika naravna, hkrati pa so za to učenje motivirani, saj z obvladovanjem jezika lažje dosežejo, kar želijo. Papert [8] razmišlja o tem, kako ustvariti okolje, v katerem bo učenje programiranja – ali učenje matematike ali katerekoli druge discipline – podobno učenju jezikov.

Papertov pristop temelji na tem, da ustvarimo okolje, v katerem se otrok uči določenih spretnosti ali veščin zato, ker želi izvesti nek projekt ali rešiti izzivalen problem. Primer je na primer risanje z želvo, pri čemer se otrok mimogrede nauči merjenja kotov. Papert je videl velik pomen računalnika v izobraževanju predvsem v tem, da nam pomaga ustvariti takšna okolja – ne pa z imitiranjem tega, kar sicer dela učitelj, ali z avtomatizacijo testov.

Kot matematika je Paperta zanimalo predvsem poučevanje matematike in programiranja. Pri tem se mu je zdelo posebno pomembno iskanje napak. Medtem ko šolski sistem v napakah pogosto vidi nekaj slabega, zaradi česar poskušajo učenci napako čimprej pobrisati in pozabiti, je Papert prepričan, da je prav iskanje in odpravljanje napak ena najpomembnejših lekcij v programiranju, ki jo lahko prenesemo tudi na vsa druga področja življenja. Kot primer navaja učence, ki so hitreje obvladali telovadne veščine, ker so o napakah, ki so jih naredili, razmišljali na podoben način kot o napakah pri

programiranju.

Pri tem je opazil tudi, da postanejo mnogi učenci, ki nimajo veselja do nekega predmeta in pri njem niso uspešni, ob takšni metodi učenja veliko uspešnejši in lahko tudi vzljubijo predmet, saj imajo stvari, ki se jih učijo, nenadoma smisel.

V okviru te diplomske naloge se ukvarjam s projekti za učenje programiranja. Okolje Scratch je zelo praktično za zgodnje učenje programiranja v prvih razredih osnovne šole [6], za starejše učence (npr. v tretjem triletju osnovnih šol) pa, kot kažejo izkušnje, ni več tako zanimivo. Ena od možnosti, ki se ponujajo v zadnjem času, je programiranje mikrokontrolerjev ali preprostih računalnikov, kot je Raspberry Pi, na katere lahko priključimo različne senzorje in naprave, kot so zvočniki, LED diode in podobno. Pričakovana prednost pred učenjem v slogu klasičnih tečajev je v tem, da je programiranje teh naprav oprijemljivejše in lažje vodi v projekte, ki zanimajo otroke.

Na osnovi izkušenj, pridobljenih na Osnovni šoli Alojzija Šuštarja, kjer so učenci programirali mikrokontroler Arduino, sem sestavila zbirko nalog za učenje programskega jezika Python z Raspberry Pi. Naloge so razdeljene v tri sklope. V prvih dveh sklopih se otrok spozna s programiranjem v Pythonu, povezanim z elektrotehniko ter programiranjem 3D sveta v Minecraftu. V zadnjem sklopu obnovi in poveže pridobljena znanja iz prvih dveh sklopov. Naloge ne zahtevajo predznanja, vendar pa se med sabo povezujejo ter stopnjujejo po težavnosti in je zato priporočljivo začeti na začetku.

Naloge so namenjene predvsem osnovnošolskim otrokom, ki se želijo spoznati z okoljem Raspberry Pi ter se pri tem naučiti programiranja v Pythonu. Naloge so predstavljene in razložene po korakih, dodatnim nalogam, ki vsebujejo namige ter razlage ostalih programskih konstruktov in funkcij, pa so dodane tudi rešitve.

Poglavje 2

Raspberry Pi

Raspberry Pi je računalnik, ki stane od 5 do 35 ameriških dolarjev, po računski moči pa se primerja z osebnimi računalniki s sredine devetdesetih. Za potrebe diplomske naloge je pomembno, da lahko na njem poganjamo Python. Za poučevanje je praktičen tudi zato, ker je ves računalnik na eni sami ploščici tiskanega vezja, ki ima ob straneh nožice ali pine, na katere lahko priključimo različne tipke in druge senzorje, LED diode, pa tudi krmilnike za motorje. Raspberry Pi lahko zato uporabljamo tudi kot mikrokontroler, obenem pa ga je mogoče programirati v višjenivojskih jezikih.

Ideja o Raspberry Piju se je razvila leta 2006 v Cambridgeu. Takrat je skupina zaposlenih na Univerzi v Cambridgeu ugotovila, da so bili novi študenti slabo pripravljeni na računalniške predmete. Večina študentov prejšnjega desetletja je bila precej bolj izkušena na področju elektrotehnike, saj jim je to bil hobi, s katerim so se pred prihodom na fakulteto ukvarjali doma. Slabo pripravljenost študentov za določene predmete so pripisali modernim prenosnikom, ki so nadomestili Sinclair Spectrum in BBC Micro iz leta 1980. Prenosniki za svojo obliko ter uporabniško prijaznim izgledom skrivajo strojno opremo, s katero mlajši študenti niso prišli v stik. Cenejša naprava bi bila za kaj takega, kot je raziskovanje računalniškega sistema, primernejša kot dragi prenosni računalniki. Zato so želeli ustvariti cenejši in manjši računalnik, ki bi zamenjal drage šolske računalnike in postal predmet, na katerem bi se

študenti in otroci lahko učili programirati. Po številnih prototipih so leta 2011 z uporabo Broadcomovega sistema na čipu (system-on-chip, SoC) ter programske opreme, ki je temeljila na operacijskemu sistemu Linux, sestavili prvo verzijo, ki so jo leto kasneje začeli prodajati. [1]

Do leta 2015 so prodali 5 milijonov računalnikov[1], med katerimi so jih največ prodali starejšim, ki jih uporabljajo pri različnih projektih. [2] V namene izobraževanja, izdelave projektov ter ostalih stvari, ki jih računalnik omogoča, so leta 2008 ustanovili fundacijo Raspberry Pi. Fundacija daje velik poudarek učenju ter poučevanju z Raspberry Pijem. [3]

Raspberry Pi se je čez leta vizualno le malo spreminjal. Njihov načrt je bil bolj kot na izboljšanje strojne opreme osredotočen na izboljšanje programske opreme. Strojno opremo so izboljševali v skladu z željami kupcev. Razpredelnica 2.1 prikazuje spremembe po letih.

Za namene diplomske naloge smo uporabljali Raspberry Pi B+, ki ima 4 vhode USB, micro SD ter 40 nožic GPIO. Ustrezal pa bo katerikoli Raspberry Pi, ki ima vsaj dva vhoda USB (za tipkovnico in miško) ter priključek HDMI za zaslon.

Raspberry Pi se od drugih računalnikov razlikuje po tem, da sestoji iz majhne ploščice tiskanega vezja, ki nima zaščite. Glavni del ploščice, ki je sestavni del vseh modelov, predstavlja sistem na čipu (system-on-chip, SoC), ki vsebuje CPU, katere hitrost je odvisna od modela in sega od 700 MHz do 1,2 GHz, GPU (VideoCore), digitalni signalni procesor in delovni pomnilnik. Njegova velikost, ki je prav tako odvisna od modela Raspberry Pi, pa je nekje med 256 MB in 1 GB. Na kartici SD (SDHC ali MicroSDHC) so shranjeni operacijski sistem, aplikacije ter ostale datoteke. Raspberry Pi se od ostalih računalnikov razlikuje tudi po tem, da ima večino spomina shranjenega na kartici SD. Prednost tega je predvsem lažja prenosljivost podatkov. Večina ploščic ima od enega vhoda do štirih vhodov USB, priključek HDMI in 3,5 mm vtikač za zvok. [10, 11]

Procesor, ki ga uporablja Raspberry Pi, se lahko primerja s čipom iPhona

[1]Eben Upton, eden od takratne skupine zaposlenih: 'Če bi mi rekli, da jih bom prodal 10.000, bi rekel, da je to neumnost.'

Leto	Model	Opis
2012	B	512 MB RAM 2 vhoda USB vhod Ethernet
2013	A	256 MB RAM 1 vhod USB nima vhoda Ethernet manjša energijska poraba kot model B lažji in cenejši kot model B namenjen uporabi na projektih
2014	B+	nadomesti model B 40 nožic GPIO 4 vhodi USB micro SD manjša energijska poraba kot model B boljši zvok kot model B
2014	A+	nadomesti model A 40 nožic GPIO manjša energijska poraba kot model A boljši zvok kot model A 2 cm krajši kot model A namenjen uporabi na projektih
2015	2 model B	nadomesti model B+ 1 GB RAM zmogljivejši procesor kot model B+ ostale lastnosti enake prejšnjemu modelu
2015	Zero	enojedrni procesor 512 MB RAM mini vhod USB mini vhod HDMI micro USB dvakrat manjši kot model A+ vendar dvakrat zmogljivejši kot model A+ namenjen uporabi na projektih
2016	3 model B	nadomesti 2 model B zmogljivejši štirijedrni procesor Bluetooth Wireless LAN ostale lasnosti enake prejšnjemu modelu namenjen splošni uporabi ali uporabi v šolah

Tabela 2.1: Razvoj modelov Raspberry Pi

3G (2008) in iPhona 3GS (2009).

V primerjavi z enojedrnim procesorjem Raspberry Pi modela B+ imata Raspberry Pi 2 (ARM Cortex-A7) in Raspberry Pi 3 (Cortex-A54) štirijedrni procesor. Čas zagona se pri obeh novejših modelih skrajša za eno tretjino. Raspberry Pi 3 je skoraj trikrat hitrejši kot Raspberry Pi B+ (glede na hitrost CPU z enojedrnim procesorjem) ter 60 odstotokov hitrejši kot Raspberry Pi 2 (glede na hitrost CPU s štirijedrnim procesorjem). [5]

Fundacija Raspberry Pi podpira inštalacijo operacijskega sistema Raspbian, ki temelji na distribuciji operacijskega sistema Linux. Raspian omogoča uporabo programskih jezikov Python in Scratch.

Poglavje 3

Python

Python je programski jezik, ki je bil ustvarjen leta 1990. Je skriptni, predmetno usmerjen jezik, ki ima dinamične podatkovne tipke in se ga zaradi počasnejšega izvajanja uporablja tudi v povezavi z drugimi programskimi jeziki, npr. C, C++, Java itd. Je višjenivojski jezik, njegovo prevajanje se izvaja sproti.

Python v primerjavi z ostalimi višjenivojskimi programskimi jeziki uporablja manj sintaktičnih konstruktov. Prav tako ne uporablja ločil in drugih znakov, kjer jih uporabljajo ostali. Ključne besede so v angleškem jeziku, napisana koda je enostavna za uporabo in vzdrževanje.

Za učenje programiranja je primeren predvsem z vidika enostavnosti. Njegovo razvojno okolje IDLE omogoča izvajanje in testiranje delov programske kode, grafični moduli, kot sta na primer *turtle* in *tkinter*, pa izboljšajo učni proces. [4]

V povezavi z Raspberry Pi je Python primeren za učenje elektrotehnike. Med iskanjem in sestavljanjem primernih nalog, ki bi bile otrokom zanimive, sem ugotovila, da je mogoče v programskem jeziku Python programirati igro Minecraft. Z vključitvijo naprav, ki jih povežemo na mikrokontroler Raspberry Pi, pa dobimo veliko idej za projekte, ki bi zanimali otroke.

Pri programskih nalogah smo se posluževali Pythonovega paketa *gpiozero*. *gpiozero* je enostaven vmesnik, ki je bil ustvarjen za lažje in enostavnejše

začetke s programiranjem mikrokontrolerja Raspberry Pi. Predhodnica knjižnice *gpiozero* je knjižnica *RPi.GPIO*, ki so jo začeli razvijati leta 2012[1]. Knjižnica omogoča nastavljanje nožic kot izhodnih ali vhodnih in kasneje kontroliranje izhodnih nožic s *high* in *low* ter branje stanja vhodnih nožic. Prvi program za prižig ledice bi s pomočjo knjižnice *RPi.GPIO* izgledal tako:

```
import RPI.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

led = 17
GPIO.setup(led, GPIO.OUT)

while True:
        GPIO.output(led, GPIO.HIGH)
        sleep(1)
        GPIO.output(led, GPIO.LOW)
        sleep(1)
```

Kljub temu da je imela knjižnica velik vpliv na izobraževanje, pa ima za programerje začetnike nekaj slabših lastnosti. Med njimi sta vključevanje preimenovanja ter uporaba nožic na dva načina. Lahko jih ločimo po lokaciji oz. zaporedni številki, lahko pa uporabljamo oznake njihovih nožic GPIO. Označevanje določimo na začetku vsakega programa, saj noben od njiju ni že vnaprej nastavljen. Prav tako bi nas program opozoril, da smo isto nožico uporabili dvakrat, če ne bi v drugi vrstici izključili opozoril. Učitelj bi se najbrž razlagi teh stvari izognil in nadaljeval s pomembnejšim delom programa. Kjub temu da knjižnico še danes uporabljajo pri številnih projektih, saj omogoča programiranje senzorjev ter ostalih naprav, pa je glavni problem, da je to le odvečna začetna koda, s katero se programerju začetniku ne bi bilo potrebno ukvarjati. Prvi program bi namesto enajstih vrstic kode moral biti krajši in ne bi smel vsebovati kode, ki je otroku ne moremo razložiti. Začetki s programiranjem v Pythonu iz ničle ali pa kot nadaljevanje iz programskega jezika (npr. Scratcha) bi morali biti dosegljivejši. [7]

V knjižnici *gpiozero* so odpravili vse nepotrebne stvari za programerja

[1]Za pomoč pri procesu izdelave piva.

začetnika. Zgornja koda, zapisana v knjižnici *gpiozero*:

```
from gpiozero import LED
from time import sleep

led = LED(17)

while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

ali pa celo samo:

```
from gpiozero import LED

led = LED(17)

led.blink()
```

Knjižnica omogoča lažje in razumljivejše začetke s programiranjem ter trenutno podpira naslednje komponente:

- LED
- RGB LED
- tipko
- motor
- napravo za oddajanje zvoka
- senzor za premikanje
- senzor za svetlobo
- robota

V drugem sklopu programskih nalog se spoznamo z Minecraftom in knjižnico *mcpi*. Minecraft Pi ima programske knjižnice za Javo in Python, ampak ne za Python 3. Knjižnica *mcpi* omogoča operacijskemu sistemu Raspbian povezavo med programskim jezikom Python in igro Minecraft. Je še en način, kako se otrok skozi igro nauči programiranja.

Spodnji program prikazuje enostavno uporabo Minecrafta v Pythonu.

```
from mcpi.minecraft import Minecraft
from gpiozero import LED
from time import sleep

mc = Minecraft.create()
led = LED(17)

while True:
        x, y, z = mc.player.getPos()
        if x == -42.7 and y == 19.0 and z == 45.5:
                led.blink()
                sleep(4)
                mc.player.setPos(x, y+50, z)
```

Picamera je Pythonov vmesnik za Raspberryjevo kamero. Omogoča zajem slik, snemanje video posnetkov, uporabo efektov itd. V novejših verzijah operacijskega sistema Raspbian je paket že vključen.

Z vidika poučevanja je pomembno, da lahko s kamero in Minecraftom uporabljamo tudi diode, tipke ter ostale stvari, ki jih lahko priključimo na Raspberry Pi. Zabavni projekti, ki se razvijejo z medsebojno uporabo knjižnic *gpiozero*, *mcpi* ter *picamera*, dajejo širši pogled na programiranje, hkrati pa dovoljujejo otrokovi domišljiji prosto pot.

Poglavje 4

Zbirka nalog

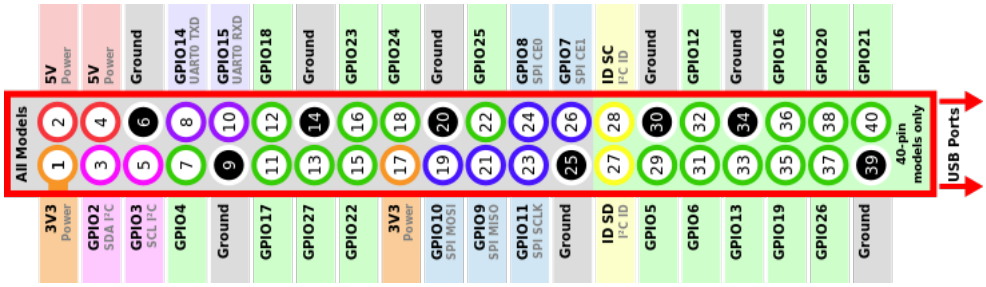
Zbirka nalog naslavlja otroke, zato je namenoma pisana v poljudnem (ne pa strokovnem) jeziku, ki je za otroke ustreznejši. Pri sestavi zbirke nalog sem si pomagala z nalogami in projekti fundacije Raspberry Pi [3] ter knjigo Adventures in Raspberry Pi [9].

4.1 Raspberry Pi

Pri nalogah bomo poleg Raspberryja potrebovali tudi diode, tipke, upore, povezovalno ploščo ter žice. Preden se lotimo povezovanja teh stvari z Raspberryjem, pa si bomo pobližje pogledali vsako od njih.

LED dioda je neke vrste majhna žarnica, ki sveti le, ko skozi njo teče tok. Tok lahko skozi njo teče le v eni smeri, in sicer od anode proti katodi. Če si diodo bolje pogledamo, lahko opazimo, da ima eno nogico krajšo od druge. Krajšo nogico imenujemo anoda, daljšo pa katoda. Tok skozi njo ustvarimo, če jo priključimo na Raspberry Pi, ki je priključen v elektriko. Raspberry Pi ima nožice imenovane tudi pini, preko katerih nanj povežemo različne stvari. Najprej bomo povezali diodo. Slika nožic 4.1, s katero si lahko pomagamo, velja za vse modele.

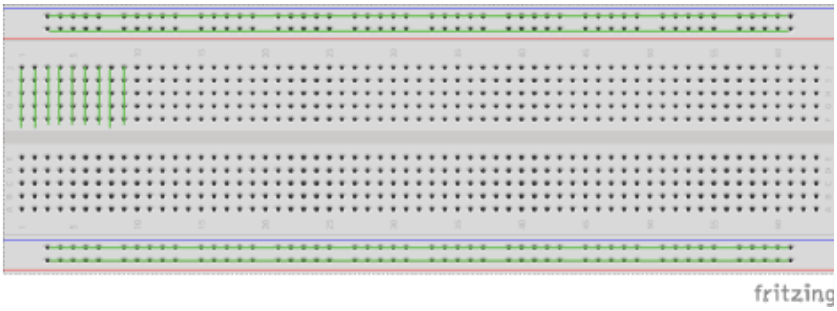
Diodo lahko na Raspberry Pi povežemo le z žicami, vendar pa nam pri kompleksnejših vezavah lahko zmanjka nožic (predvsem nožic GND). V izogib



Slika 4.1: Oznake nožic

temu uporabimo povezovalno ploščo na sliki 4.2, ki omogoča množenje nožic.

V luknje povezovalne plošče vstavimo stvari, ki jih želimo povezati z Raspberryjem. Luknjice pa so med sabo povezane tako, kot kažejo zelene črte na sliki.



Slika 4.2: Povezovalna plošča

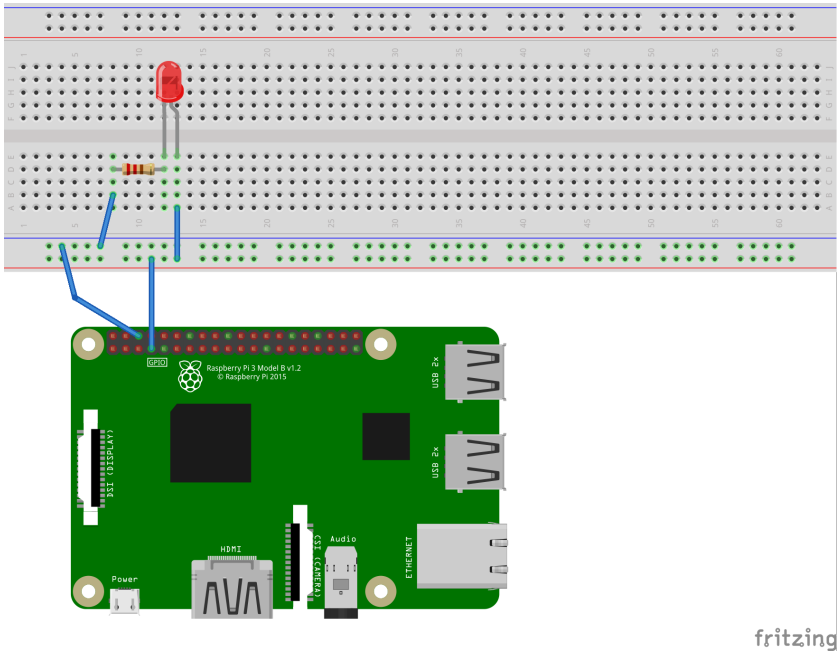
Če na rdečo črto povežemo Raspberryjevo nožico GND, bomo do njega lahko dostopali na celotni črti. Prav tako lahko storimo z nožico +5V in +3V.

Za priključitev LED diode bomo potrebovali GND in +5V. Nekaj podobnega lahko poskušaš narediti z baterijo. Če boš LED diodo pravilno obrnil, bo zasvetila. Takrat bo namreč krog sklenjen. Mi bomo namesto baterije uporabljali Raspberry Pi.

Da pa skozi diodo ne steče preveliko elektrike, ki jo lahko poškoduje, imamo še eno stvar, imenovano upor. Upori, ki jih bomo potrebovali pri diodah, imajo približno 300 ohmov. Upor nastavimo na povezavo med diodo in GND.

Prižig diode

Diodo poveži z Raspberryjem.



Slika 4.3: Vezje z eno diodo

Diodo smo povezali na nožico GPIO zato, da jo bomo lahko 'nadzorovali'. Pošiljali ji bomo ukaze, kot na primer: prižgi se, ugasni se, utripaj ... Vse to ne bi bilo mogoče, če bi jo povezali na +5V. Slika 4.3 prikazuje končno vezje.

Cilji:

- vključitev knjižnice *gpiozero* in modula *time*
- prižgati in ugasniti diodo
- uporaba ukaza *sleep*
- utripanje z diodo s pomočjo fukcije *blink*()
- definicija spremenljivke

Potek:

1. V Meniju pod Programming poiščemo Python 2 (IDLE) ter kliknemo nanj.

> Za delo s Pythonom bomo uporabljali programsko okolje IDLE. Ta ima možnost ustvarjanja in urejanja kode v programskem jeziku Python. Okolje omogoča preverjanje napak programov, napisanih v Pythonu.

Kliknemo na gumb File in potem New File, da ustvarimo novo datoteko. Datoteko shranimo in poimenujemo s klikom na File ¿ Save as. Poimenujemo jo *prizigDiode.py* ter pritisnemo enter. Končnica *.py* pomeni, da je to Pythonova datoteka. Od sedaj naprej bomo vsak program zapisali v svojo datoteko.

Na začetku programa moramo našteti vse module in knjižnice, ki jih Raspberry Pi potrebuje pri delu z diodami. Za delo z bolj sprecifičnimi stvarmi bomo potrebovali pomoč nekaterih Pythonovih knjižnic.

Knjižnica *gpiozero* je namenjena upravljanju diod, branju tipk in podobno. Vezje smo sestavili tako, da smo pozitivni del diode povezali z nožico GPIO4. Sedaj bomo s pomočjo knjižnice *gpiozero* to povedali tudi Raspberry Piju.

```
from gpiozero import LED
from time import sleep
```

> Kasneje boš opazil, da pri programih velikokrat uporabljamo iste dele kode. Veliko programskih jezikov se ponavljanju pisanja kode izogne tako, da te dele zapakira v pakete, imenovane moduli. Python ima veliko število modulov, ki vsebujejo uporabno kodo, ki jo lahko večkrat uporabimo. Modul *time* omogoča uporabo določenih funkcij, povezanih s časom.

2. Ustvarimo spremenljivko led ter ji povemo, na katero nožico smo povezali diodo.

```
led = LED(4)
```

Spremenljivke si lahko predstavljamo kot škatle, v katere nekaj shranimo. Škatle oz. spremenljivke med sabo ločujemo po imenih. Na primer:

```
i = 1
a = 8
d = 3
```

Vanje pa poleg števil lahko shranjujemo tudi besedilo. Več o njih bomo izvedeli kasneje. Zaenkrat je dovolj le, da vemo, da nam $LED(4)$ pove, da je na nožici 4 ledica ali dioda. Ker smo to shranili v spremenljivko led, jo bomo prižigali in ugašali le z uporabo te spremenljivke.

Diodo prižgemo s pomočjo funkcije $on()$.

> Funkcija je del kode, ki pove računalniku, naj izvede določeno nalogo. Uporabimo jo lahko večkrat.

```
led.on()
```

Ugasnemo pa jo s pomočjo funkcije $off()$:

```
led.off()
```

Če angleškega izraza on in off še ne poznaš, sedaj veš, kaj pomenita. Najdeš ju na skoraj vsaki napravi.

3. Če hočemo, da dioda sveti 3 sekunde, uporabimo funkcijo $sleep()$ iz modula *time*. Številki v oklepaju, ki predstavlja čas v sekundah, pravimo argument.

> Argument je del informacije, ki jo funkcija potrebuje za svojo izvedbo. Argument vedno zapišemo v oklepaj, ki sledi imenu funkcije.

```
led.on()
sleep(3)
```

Dioda se prižge in zaspi za naslednje 3 sekunde. Vendar ne zares; v resnici Raspberry Pi 3 sekunde ne počne čisto ničesar drugega, kot čaka, da te 3 sekunde minejo. Zato se zdi, da s tem uspavamo diodo.

Sedaj ko se znaš pogovarjati z Raspberryjem, mu povej, naj prižge diodo, počaka 5 sekund ter jo ugasne.

Tvoja koda naj bi izgledala nekako tako:

```
from gpiozero import LED
from time import sleep

led = LED(4)
led.on()
sleep(5)
led.off()
```

Sedaj pa poizkusi namesto:

```
sleep(5)
```

napisati:

```
#sleep(5)
```

Kaj se zgodi z diodo? Kaj pomeni znak #?

Ta znak pred ukazom pomeni, da Raspberry Piju sporočamo, naj pozabi celotno vrstico in naj je ne prebere. V našem primeru torej dioda ne sveti 5 sekund, temveč v istem trenutku tudi ugasne. Ker se to zgodi tako hitro, se ti zdi, da se dioda sploh ne prižge. Funkcijo *sleep*() bomo od sedaj naprej ravno zato kar pogosto uporabljali.

Komentarje uporabimo v kodi, če si želimo zapisati, kaj kateri del kode naredi. Komentar je torej opazka, ki se začne s simbolom #. Ta sporoči prevajalniku, naj vrstico za njim ignorira. Če se komentar razteza čez več vrstic, moramo simbol # zapisati na začetku vsake vrstice. Uporaba komentarjev je koristna tudi v šoli, če tvojo kodo bere sošolec ali pa učitelj. Komentarji so jim v pomoč pri razumevanju kode.

```
from gpiozero import LED
from time import sleep

#program ne bo storil nicesar ker je tole
#le opazka zame
```

Dodatne naloge:

1. Trikat prižgi in ugasni isto diodo.

2. Razmisli, kaj bi bilo potrebno dodati, da bi dioda utripala, torej da bi se prižgala in ugasnila več kot petkrat.

Funkcija *blink*() omogoča utripanje diode.

Uporaba:

```
led.blink()
```

Rešitvi: 1.

```
from gpiozero import LED
from time import sleep

led = LED(4)

led.on()          #prvic
sleep(1)
led.off()
sleep(1)

led.on()          #drugic
sleep(1)
led.off()
sleep(1)

led.on()          #tretjic
sleep(1)
led.off()
sleep(1)
```

2. Učinek utripanja nam da spodnji program, kjer uporabimo funkcijo *blink*().

```
from gpiozero import LED

led = LED(4)

led.blink()
```

Program ustaviš s kombinacijo tipk Ctrl + C.

V računalništvu za ponavljanje stvari uporabljamo zanke. Zgornji program lahko brez uporabe funkcije *blink*() zapišemo tako:

```
from gpiozero import LED
```

```
from time import sleep

led = LED(4)

while True:
        led.on()
        sleep(1)
        led.off()
        sleep(1)
```

Preizkusi, če dioda utripa. Če utripa počasneje ali hitreje, je to zaradi argumenta, ki ga podamo funkciji *sleep*().

Zaenkrat si zapomnimo le, da stvari, ki jih želimo ponavljati, zamaknemo, nad njimi pa napišemo *while True* :. Kaj je *while* in kaj pomeni *True*, bomo izvedeli malo kasneje.

Tipka in dioda

Cilji:

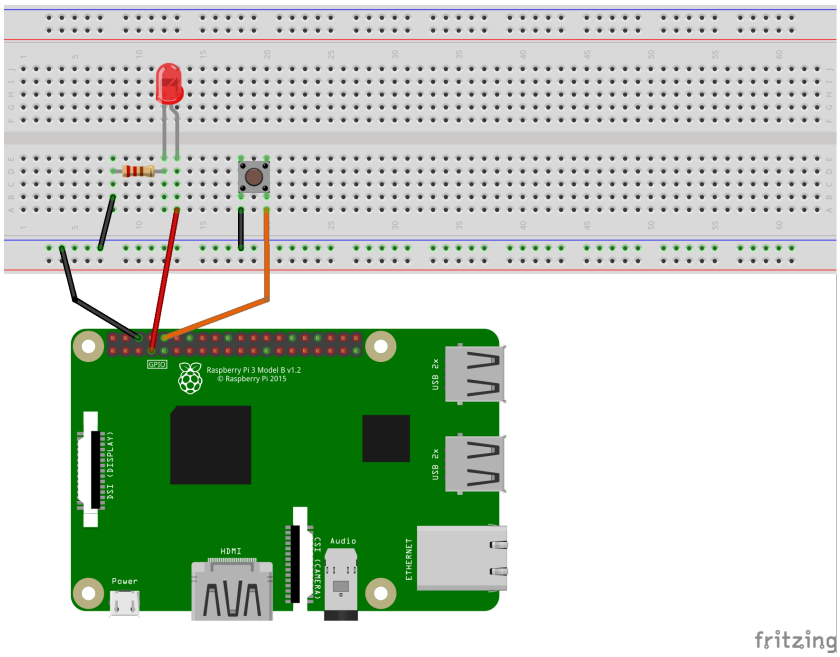
- vezava tipke
- razumevanje in uporaba osnovnih funkcij iz modula *Button*
- uporaba stavka *if* in *else*
- s pomočjo funkcije *print*() izpisati besedilo
- uporaba funkcije *toggle*()
- uporaba zanke *while*
- definiranje funkcije in njena uporaba

Potek:

1. Podobno kot za LED diodo tudi za tipko potrebujemo modul. Oba modula pa vključujemo iz iste knjižnice (*gpiozero*) zato lahko zapišemo tako:

```
from gpiozero import LED, Button
```

2. Tipko bomo uporabili za prižig diode. V trenutno vezje z eno diodo dodaj tipko tako, da je z eno stranjo povezana na GND, z drugo pa na prost GPIO izhod. Končno vezje je predstavljeno na sliki 4.4.



Slika 4.4: Vezje s tipko in diodo

3. Uporabimo že pridobljeno znanje in zapišemo:

```
led = LED(4)
tipka = Button(15)

led.on()
sleep(1)
led.off()
```

Podobno kot podamo Raspberry Piju število nožice, na katerega je povezana dioda, podamo tudi število nožice tipke. Ustvarimo spremenljivko *tipka*.

Shrani in preizkusi svoj program. Če ne dela, preveri, kako si na svojem vezju povezal tipko. Če je povezan s kakšno drugo GPIO nožico, namesto 15 zapiši drugo število.

Vendar pa tipka še vedno ne bo delovala. Manjka nam namreč odziv na tipko. Hočemo, da dioda posveti, ko bomo pritisnili na tipko. Torej bomo

pred *led.on*() napisali sledeče:

```
tipka.wait_for_press()
```

Tale zadeva čaka, da pritisneš na tipko, in šele potem izvede stvari, napisane za njo.

Končen program izgleda tako:

```
from gpiozero import LED, Button
from time import sleep

led = LED(4)
tipka = Button(15)

tipka.wait_for_press()
led.on()
sleep(1)
led.off()
```

Shrani in ga preizkusi.

Dodatne naloge:

1. Preveri, če je tipka pritisnjena, in to tudi izpiši na zaslon.

   Pri preverjanju, če je tipka pritisnjena, bomo potrebovali pogoje. Ustvarjanje pogoja je podobno kot postavljanje vprašanja, na katerega lahko odgovorimo z dvema ali več odgovori. Vprašaš lahko na primer: Ali dežuje? Če je odgovor da, potem bi moral vzeti dežnik; če je odgovor ne, potem ne potrebuješ jakne. Ključna beseda tukaj je če. V angleščini se uporablja beseda if. Pogoj je izpolnjen, kadar na vprašanje odgovorimo z da, in ni izpolnjen, kadar na vprašanje odgovorimo z ne.

   if in *if...else* sta zelo pogosta konstrukta programiranja. Če uporabimo stavek *if*, sprašujemo, če je pogoj za njim izpolnjen. Če je izpolnjen, želimo nekaj storiti. Na primer: Če dežuje, potem vzemi dežnik. Lahko pa dodamo tudi del, kjer preverjamo, če pogoj ni izpolnjen. To zapišemo s stavkom *else*. Na primer: Če dežuje, potem vzemi dežnik; drugače pa vzemi očala.

Primer:

```
if 2>1:
        print("2 je res vec kot 1")
else:
        print("Moj Raspberry Pi")
```

Primer izpiše stavek *'2 je res več kot 1'*, zato ker je pogoj vedno resničen.

Za izpis teksta na zaslon potrebujemo funkcijo *print*(). Funkcija *print*() je le ena izmed standardnih funkcij, ki jih računalnik že razume. Kot argument ji podamo niz oz. besedilo, ki želimo, da ga izpiše. Beseda *String* je angleška beseda za niz in predstavlja informacijo, ki jo vnesemo kot besedilo. Besedilo zapišemo v dvojnih narekovajih, na ta način računalnik ve, da gre za niz.

Funkcija *is_pressed*() preverja, če je tipka pritisnjena.

Uporaba:

```
if tipka.is_pressed:
        print("Tipka je bila pritisnjena")
```

2. Razmisli, kako bi tvoja tipka lahko postala stikalo. Stikalo stalno preverja, če je tipka pritisnjena. Dioda naj se ob pritisku tipke prižge, če je bila prej ugasnjena, in ugasne, če je prej svetila.

Funkcija *toggle*() spremeni stanje diode.

Uporaba:

```
led.toggle()
```

Poskusi se spomniti, kaj smo poleg funkcije *blink*() uporabili za utripanje diode. Prav tako kot pri utripanju z diodo bomo tudi tukaj potrebovali ponavljanje. Ponavljali bomo vprašanje: Ali je tipka pri-

tisnjena? Za to bomo uporabili stavek *if*. Za ponavljanje stavka *if* pa bomo uporabili stavek *while*. *While* prav tako kot *if* potrebuje pogoj. Če je pogoj izpolnjen oz. resničen, potem se bodo izvedli ukazi, ki smo jih zamaknili. Za razliko od stavka *if* pa se bo stavek *while* izvedel ponovno, če je pogoj še vedno izpolnjen. Če zapišemo pogoj, ki bo vedno izpolnjen, se bo koda, ki smo jo zamaknili, izvajala, dokler programa ne prekinemo. Tak pogoj je *True*. *True* je torej pogoj, ki je vedno resničen oz. izpolnjen. *While* pa je zanka, ki izvaja kodo, dokler je pogoj, napisan za njo, resničen.

Primer:

```
while 2>1:
        print("Moj␣Raspberry␣Pi")
```

Tale zanka neskončnokrat izpiše stavek *'Moj Raspberry Pi'*, saj je 2 vedno več kot 1.

3. Sedaj pa poskusi napisati program, ki bo rekel diodi, naj sveti vsakič, ko bo tipka pritisnjena.

   Namig: Uporabi zanko *while*.

Rešitve:

1.

```
from gpiozero import Button

tipka = Button(15)

if tipka.is_pressed:
        print("Pritisnil␣si␣na␣tipko!")
else:
        print("Tipka␣ni␣bila␣pritisnjena")
```

Če program poženemo in pritisnemo na tipko, opazimo, da se bo na zaslon izpisalo besedilo *'Pritisnil si na tipko'*. Podobno kot tiskalnik lahko tudi mi 'printamo' stvari. Printamo besedilo, ki je v oklepaju za funkcijo *print*().

2. Funkcija *toggle*() torej vsakič spremeni stanje diode. Če je prižgana, jo izklopi in obratno.

```
from gpiozero import LED, Button

led = LED(4)
tipka = Button(15)

while True:
        tipka.wait_for_press()
        led.toggle()
```

Zgornji program v zanki preverja in čaka, da je tipka pritisnjena. Nato sledi sprememba stanja diode. Medtem pa spodnji program namesto neskončne zanke uporabi funkcijo *when_pressed*(). Ta že sama po sebi stalno preverja, če je tipka pritisnjena. Vsakič ko je, izvede funkcijo *toggle*().

```
from gpiozero import LED, Button

led = LED(4)
tipka = Button(15)

tipka.when_pressed = led.toggle
```

3. Ideja je zelo podobna ideji prve dodatne naloge, ki je: preveri, če je tipka pritisnjena; če je, naj dioda posveti. Ker pa hočemo, da se to dogaja ves čas, bomo uporabili zanko.

```
from gpiozero import LED, Button

led = LED(4)
tipka = Button(15)

while True:
        if tipka.is_pressed:
                led.on()
        else:
                led.off()
```

Shrani in preveri, če deluje.

Lahko pa uporabimo tudi funkcijo *when_pressed*(). V prejšnji nalogi smo videli, da pritisk na tipko omogoči izvedbo neke funkcije. Kot že vemo, je funkcija del kode, ki jo lahko uporabimo večkrat. Zaenkrat smo bili navajeni uporabljati že ustvarjene funkcije iz modulov, sedaj pa se bomo naučili

napisati svojo. Pisanje funkcije je precej enostavno. Tole je primer funkcije, ki ne naredi nič kaj koristnega.

```
def tiskaj():
        print("Moj␣Raspberry␣Pi")
```

Funkcija, ki jo želimo uporabiti pri nalogi, pa mora prižgati in ugasniti diodo. To že znamo, zato zapišemo funkcijo $PrizgiUgasni()$:

```
def prizgiUgasni():
        led.on()
        sleep(1)
        led.off()
```

Če želimo, da se funkcija izvede vsakič, ko bo pritisnjena tipka, pa zapišemo:

```
from gpiozero import LED, Button
from time import sleep

led = LED(4)
tipka = Button(15)

def prizgiUgasni():
        led.on()
        sleep(1)
        led.off()

tipka.when_pressed = prizgiUgasni
```

Prižgi vseh 7

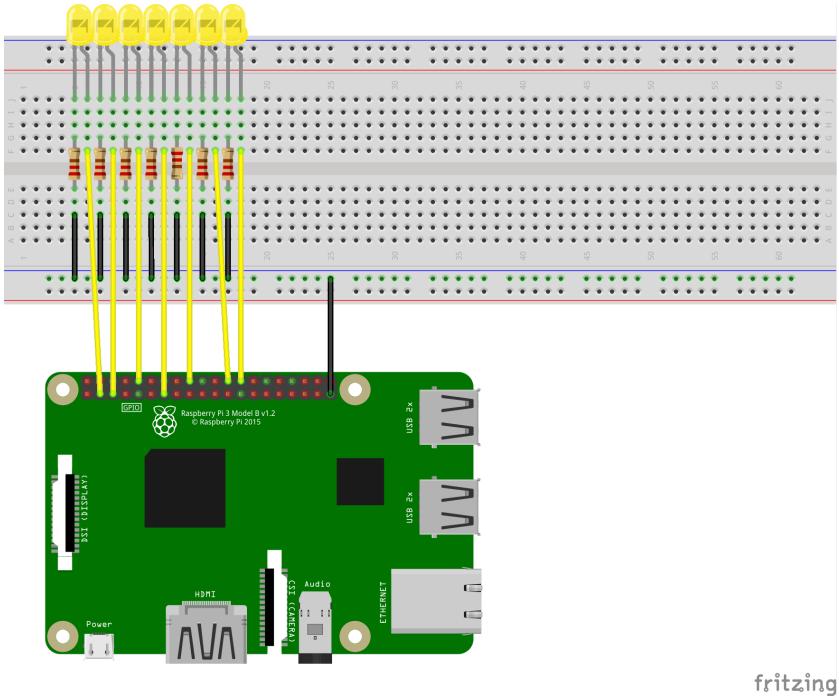
Cilji:

- vezava več LED diod
- kaj je seznam in njegova uporaba
- uporaba zanke for

Potek:

1. Najprej sestavi vezje 7 diod, postavljenih eno za drugo. Vezje je predstavljeno na sliki 4.5.

2. Odpri *Python 2 (IDLE)*, ustvari novo datoteko ter vključi knjižnico, ki jo potrebuješ za delo z diodami.



Slika 4.5: Vezje s 7 diodami

```
from gpiozero import LED
```

3. Nato vsaki diodi povej, na katero nožico je povezana, pri tem pazi, da navedeš pravilne številke. Ko to uspešno storiš, diodo prižgi eno za drugo.

```
led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

led1.on()
sleep(1)
led2.on()
sleep(1)
led3.on()
sleep(1)
led4.on()
sleep(1)
led5.on()
sleep(1)
led6.on()
```

```
sleep(1)
led7.on()
```

Dodatne naloge:

1. Prižgi in ugasni prvo diodo, nato drugo, potem vse do zadnje.

   Ponavljanje stvari je v računalništvu zelo pogosto, saj lahko del kode napišemo le enkrat in ga nato ponovimo sedemkrat. Do sedaj smo del kode ponavljali s pomočjo zanke *while*. Poleg zanke *while* pa poznamo tudi zanko *for*. Odpremo datoteko in vanjo zapišemo:

   ```
   for i in [0,1,2,3]:
           print("Moj␣Raspberry␣Pi")
   ```

   Koda pravi, da za vsak i iz nabora števil $[0, 1, 2, 3]$ izpišemo stavek *'Moj Raspberry Pi'*. Kolikokrat misliš, da se bo stavek izpisal? Poženi kodo in preštej. Če v nabor števil dodaš še število 4, se bo stavek izpisal petkrat. Zaenkrat ni pomembno, katera števila zapišeš v nabor, pomembno je, koliko je teh števil. Števila iz nabora pa se bodo izpisala, če napišemo:

   ```
   for i in [0,1,2,3]:
           print(i)
   ```

   Če sedaj dodaš število 5 namesto 4, se bodi izpisala števila od 0 do 5 brez števila 4. Naboru števil rečemo seznam. Prepoznamo ga po oglatih oklepajih. Stvari znotraj seznama se imenujejo elementi, te ločimo z vejicami. V seznam lahko poleg števil shranjujemo tudi druge stvari, npr. nize.

2. Prižgi in ugasni v eno smer, pri tem vključi še prižiganje in ugašanje v drugo smer.

   > Namig: Poimenuj dva seznama ter dve funkciji.

3. Ponavljaj: Prižgi in ugasni prvo diodo, nato drugo, potem vse do zadnje (valovi naj se premikajo le v eno smer).

> Namig: Pomagaj si s prvo dodatno nalogo ter zanko *while*.

4. Ponavljaj: Prižgi in ugasni v eno smer ter prižgi in ugasni v drugo smer (valovi naj se vsakič, ko pridejo do robne diode, odbijejo in vrnejo. Pri tem naj pri odboju prva in zadnja dioda zasvetita le enkrat).

> Namig: Pomagaj si z drugo dodatno nalogo ter zanko *while*.

Rešitve:

1.

```
from gpiozero import LED
from time import sleep

led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

led1.on()
sleep(1)
led1.off()
led2.on()
sleep(1)
led2.off()
led3.on()
sleep(1)
led3.off()
led4.on()
sleep(1)
led4.off()
led5.on()
sleep(1)
led5.off()
led6.on()
sleep(1)
led6.off()
led7.on()
sleep(1)
led7.off()
```

Tole je rešitev, ki pa ni lepa. Ker želimo, da bi bila, se bomo na pomoč obrnili k Pythonovi strukturi, imenovani seznam. Poimenujmo naš seznam *seznamLedic* in v njega zapišimo imena LED diod. Zapišemo jih v pravilnem vrstnem redu, saj bo to kasneje zelo pomembno.

```
seznamLedic = [led1, led2, led3, led4, led5, led6, led7]
```

Do elementov v seznamu pa dostopamo s pomočjo zanke *for*.

```
seznamLedic = [led1, led2, led3, led4, led5, led6, led7]

for led in seznamLedic:
        led.on()
        sleep(1)
        led.off()
```

Koda pravi, da vsak *led* iz seznama prižgemo, počakamo eno sekundo in jo nato ugasnemo. Prepričaj se sam.

```
from gpiozero import LED
from time import sleep

led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

seznamLedic = [led1, led2, led3, led4, led5, led6, led7]

for led in seznamLedic:
        led.on()
        sleep(1)
        led.off()
```

Če primerjamo obe rešitvi, vidimo, da je druga krajša in zaradi tega tudi lepša. Od sedaj naprej bomo več diod prižigali na tak način.

2. Ena funkcija naj diode prižge in ugasne v eno stran, druga pa naj jih prižge in ugasne v nasprotno.

```
from gpiozero import LED
from time import sleep

led1 = LED(2)
led2 = LED(3)
```

```
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

seznamLedic1 = [led1, led2, led3, led4, led5, led6, led7]
seznamLedic2 = [led7, led6, led5, led4, led3, led2, led1]

def prizgi_ugasni_tja():
        for led in seznamLedic1:
                led.on()
                sleep(1)
                led.off()

def prizgi_ugasni_nazaj():
        for led in seznamLedic2:
                led.on()
                sleep(1)
                led.off()

prizgi_ugasni_tja()
prizgi_ugasni_nazaj()
```

Če si pozoren, lahko opaziš, da zadnja dioda sveti dvakrat dlje kot ostale. Temu je tako, ker jo dvakrat prižgemo in ugasnemo, medtem ko ostale samo enkrat.

3.

```
from gpiozero import LED
from time import sleep

led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

seznamLedic = [led1, led2, led3, led4, led5, led6, led7]

def prizgi_ugasni():
        for led in seznamLedic:
                led.on()
                sleep(1)
                led.off()
```

```
while True:
        prizgi_ugasni()
```

Funkcij ne uporabljamo samo takrat, ko ne želimo ponavljati kode, uporabljamo jih tudi takrat, kadar želimo napisati lepo in pregledno kodo. V zgornjem programu smo zato napisali funkcijo, ki prižge in ugasne diode od prve do zadnje. Valovanje ustvarimo z zanko *while*, ki funkcijo *prizgi_ugasni*() izvaja, dokler programa ne ustavimo s pritiskom na tipki *Ctrl + C*.

4.

```
from gpiozero import LED
from time import sleep

led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

seznamLedic1 = [led1, led2, led3, led4, led5, led6]
seznamLedic2 = [led7, led6, led5, led4, led3, led2]

def prizgi_ugasni_tja():
        for led in seznamLedic1:
                led.on()
                sleep(1)
                led.off()

def prizgi_ugasni_nazaj():
        for led in seznamLedic2:
                led.on()
                sleep(1)
                led.off()

while True:
        prizgi_ugasni_tja()
        prizgi_ugasni_nazaj()
```

Za ponavljanje uporabimo zanko *while*, v kateri najprej prižgemo in ugasnemo diode v eno stran, nato v drugo. Dvojnemu prižiganju se izognemo, če iz prvega seznama zbrišemo *led*7, iz drugega pa *led*1.

Tipka in valovanje

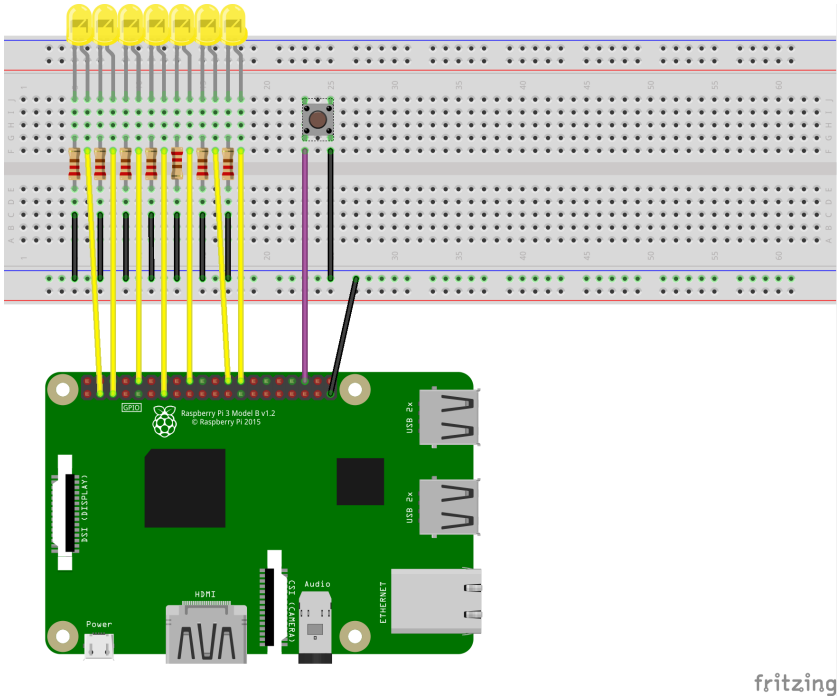
Cilji:

- razumevanje lastne kode
- postavitev pogoja na pravilno mesto
- ponovitev funkcij modula *LED* in *Button*

Ob pritisku na tipko naj bo valovanje hitrejše. Valovanje naj bo enako valovanju iz zadnje naloge prejšnjega poglavja, torej naj se odbija.

Potek:

1. Tipko dodaj na vezje, ki si ga uporabljal do sedaj. Poveži ga na *GND* ter na nožico GPIO, kot kaže na sliki 4.6.



Slika 4.6: Vezje s 7 diodami in eno tipko

2. Odpri novo datoteko, jo poimenuj in shrani. V kodo vključimo tudi tipko, tako da bo začetek kode izgledal tako:

```
from gpiozero import LED, Button
from time import sleep

tipka = Button(16)
```

3. Na hitrost valovanja vpliva čas spanja oz. čakanja. V kodi čakamo na dveh mestih. Če čas čakanja spremenimo z ene sekunde na 0,2 s, bo valovanje veliko hitrejše.

```
def prizgi_ugasni_tja():
        for led in seznamLedic1:
                led.on()
                sleep(0.2)
                led.off()

def prizgi_ugasni_nazaj():
        for led in seznamLedic2:
                led.on()
                sleep(0.2)
                led.off()
```

Preveri, shrani in poženi program.

4. Vemo torej, da hočemo imeti krajši čas spanja, če je tipka pritisnjena. Zato bomo uporabili pogoj:

```
if tipka.is_pressed:
        sleep(0.2)
else:
        sleep(1)
```

To bomo vstavili na mesto, kjer zaenkrat stoji *sleep*(1). Koda bo izgledala nekako tako:

```
from gpiozero import LED
from time import sleep

led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

seznamLedic1 = [led1, led2, led3, led4, led5, led6]
seznamLedic2 = [led7, led6, led5, led4, led3, led2]

def prizgi_ugasni_tja():
```

```
        for led in seznamLedic1:
                led.on()
                if tipka.is_pressed:
                        sleep(0.2)
                else:
                        sleep(1)
                led.off()

def prizgi_ugasni_nazaj():
        for led in seznamLedic2:
                led.on()
                if tipka.is_pressed:
                        sleep(0.2)
                else:
                        sleep(1)
                led.off()

while True:
        prizgi_ugasni_tja()
        prizgi_ugasni_nazaj()
```

Preizkusi sam. Ne pozabi, da je čas za prižiganje diod odvisen od argumenta, ki ga podaš funkciji *sleep*(). Če se ti zdi, da se prižigajo prepočasi, lahko čas čakanja zmanjšaš.

Dodatne naloge:

1. Program naj izvaja valovanje v eno stran. S pritiskom na tipko zamenjaj stran valovanja.

   Namig: Uporabi funkcijo *sleep*().

2. Naj pritisk na tipko sproži val. Poskusi ustvariti drugi val, še preden se prvi konča. Kaj opaziš?

   Namig: Pomagaj si z zanko *while* in funkcijo *is_pressed*().

Rešitvi:

1.

```
from gpiozero import LED
from time import sleep

led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

seznamLedic1 = [led1, led2, led3, led4, led5, led6]
seznamLedic2 = [led7, led6, led5, led4, led3, led2]

def prizgi_ugasni_tja():
        for led in seznamLedic1:
                led.on()
                slep(0.5)
                led.off()

def prizgi_ugasni_nazaj():
        for led in seznamLedic2:
                led.on()
                sleep(0.5)
                led.off()

while True:
        if tipka.is_pressed:
                prizgi_ugasni_nazaj()
        else:
                prizgi_ugasni_tja()
```

2.

```
from gpiozero import LED
from time import sleep

led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

seznamLedic1 = [led1, led2, led3, led4, led5, led6]
seznamLedic2 = [led7, led6, led5, led4, led3, led2,led1]

def prizgi_ugasni_tja():
        for led in seznamLedic1:
```

```
                led.on()
                sleep(1)
                led.off()

def prizgi_ugasni_nazaj():
        for led in seznamLedic2:
                led.on()
                sleep(1)
                led.off()

while True:
        if tipka.is_pressed():
                prizgi_ugasni_tja()
                prizgi_ugasni_nazaj()
```

Reakcijski čas

Cilji:

- pravilno sestaviti vezje
- uporaba funkcije $uniform()$ za naključno izbiranje
- uporaba funkcije $input()$
- boljše razumevanje spremenljivk

Na Raspberry Pi priključi dve tipki in eno diodo. Pri vezavi si lahko pomagaš s sliko 4.7.

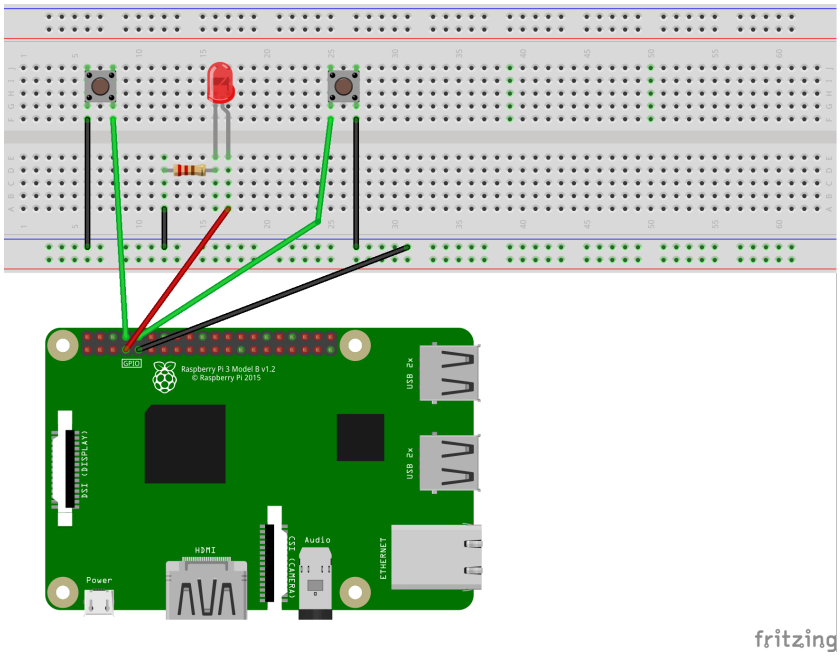
Pravila: Hkrati tekmujeta dva tekmovalca. Ko se dioda prižge, Raspberry Pi zazna, kdo od njiju je tipko pritisnil hitreje. Zmaga tisti, ki je hitrejši.

Potek:

1. Ustvari in shrani nov dokument z imenom *reakcija.py*. Nato vpelji module in knjižnice, ki jih potrebuješ za kontrolo nožic Raspberry Pi.

```
from gpiozero import LED, Button
from time import sleep
```

2. Ker bo izhod dioda, ki si jo povezal, je potrebno nastaviti ravno tisto nožico, s katerim je povezana dioda. Ustvari spremenljivko *led* in označi nožico, s katero si jo povezal. Nato počakaj 5 sekund in prižgi diodo.



Slika 4.7: Vezje z eno diodo in dvema tipkama

```
led = LED(4)

sleep(5)
led.on()
```

Dokument shrani ter ga poženi. Če se dioda po 5 sekundah ne prižge, poglej, če si se kje pri tipkanju zmotil.

3. Dodati moramo še element presenečenja. Preden prižgemo diodo, počakamo naključno dolgo. Na začetek programa zato dodamo potreben modul in njegovo funkcijo:

```
from random import uniform
```

Poišči, kje v kodi čakaš 5 sekund, ter jo preuredi tako, da bo pisalo:

```
sleep(uniform(5,10))
```

To pomeni, da bo čas čakanja na prižig diode nekje med 5 do 10 sekund. Točnega časa ne vemo, ker ga računalnik izbere naključno. Naključno število nato kot argument podamo funkciji *sleep*().

4. Sledi dodajanje tipk v kodo. Pod spremenljivko *led* dodamo dve novi spremenljivki za tipki.

```
leva_tipka = Button(14)
desna_tipka = Button(15)
```

Pri tem pazi, da številki v oklepajih ustrezata nožicama, na kateri sta priključeni dve tipki. Drugače pritisk na tipko ne bo povzročil ničesar.

Kako preverimo, če sta tipki pritisnjeni, že vemo. Sporočilo, ki ga želimo zapisati na zaslon, napišemo s pomočjo funkcije *print*().

```
if leva_tipka.is_pressed:
        print("Levi␣igralec␣je␣zmagal.")
else:
        print("Desni␣igralec␣je␣zmagal.")
```

5. Če želimo, da program namesto številke tipke izpiše ime igralca, moramo uporabiti funkcijo *input*(). Funkcija najprej izpiše besedilo, ki ga zapišemo v oklepaj, nato pa čaka, da uporabnik vnese besedilo. Besedilo, ki ga vnese, mora biti v dvojnih narekovajih, ker funkcija sprejema niz oz. besedilo. Ker nas zanimata imeni obeh igralcev, zapišemo tako:

```
levo_ime = input("Levi␣igralec␣je␣")
desno_ime = input("Desni␣igralec␣je␣")
```

Ko vpišeš ime, pritisni Enter, da potrdiš svoj vnos. V spremenljivki *levo_ime* in *desno_ime* smo sedaj shranili imeni obeh igralcev.

Spremenimo le še funkcijo, ki nam pove, kdo je zmagal, tako dobimo končen program:

```
from gpiozero import LED, Button
from time import sleep
from random import uniform

led = LED(4)
leva_tipka = Button(14)
desna_tipka = Button(15)

levo_ime = input("Levi␣igralec␣je␣")
desno_ime = input("Desni␣igralec␣je␣")

sleep(uniform(5,10))
led.on()
if leva_tipka.is_pressed:
        print(levo_ime + "␣je␣zmagal")
else:
        print(desno_ime + "␣je␣zmagal")
```

Vrednost imena, ki ga želimo izpisati, napišemo v oklepaj funkcije *print*(). Nize med sabo združujemo s plusom. Na tak način smo združili *levo_ime* ter besedilo *'je zmagal'*.

Dodatne naloge:

1. Kodo lahko damo v zanko, da se dioda prižge večkrat. Razmisli, katero zanko bi uporabil in kam bi prišla. Vključi spremenljivko *stRund*, ki predstavlja število rund, ki sta jih s sotekmovalcem pripravljena odigrati. Naj bo na začetku enaka 5, nato pa jo zmanjšuj.

   Spremenljivko za ena povečaš ali zmanjšaš tako:

   ```
   stevilo = 1
   stevilo = stevilo + 1
   ```

   Vrednost spremenljivke stevilo bo sedaj 2, ker smo sešteli 1 + 1. Zapišimo še:

   ```
   stevilo = stevilo - 1
   ```

   V tem primeru pa bo vrednost spremenljivke ponovno 1, ker smo vrednosti 2 odšteli 1.

   Krajše lahko zapišemo:

   ```
   stevilo += 1    #pristejemo
   stevilo -= 1    #odstejemo
   ```

2. Program poskusi preurediti tako, da vsak krog izpiše skupne točke obeh tekmovalcev. Dodati bo potrebno tudi to:

   ```
   print(levi_stzmag + ":" + desni_stzmag)
   ```

   Razmisli, kaj sta spremenljivki *levi_stzmag* in *desni_stzmag*, kakšni sta ob začetku igre ter kje se spreminjata.

Rešitvi:

1.

```
from gpiozero import LED, Button
from time import sleep
from random import uniform

led = LED(4)
leva_tipka = Button(14)
desna_tipka = Button(15)


levo_ime = input("Levi␣igralec␣je␣")
desno_ime = input("Desni␣igralec␣je␣")
stRund = 5

while stRund>0:
        sleep(uniform(5,10))
        led.on()
        if leva_tipka.is_pressed:
                print(levo_ime + "␣je␣zmagal")
        else:
                print(desno_ime + "␣je␣zmagal")
        stRund -= 1
```

2.

```
from gpiozero import LED, Button
from time import sleep
from random import uniform

led = LED(4)
leva_tipka = Button(14)
desna_tipka = Button(15)

levo_ime = input("levi␣igralec␣je␣")
desno_ime = input("desni␣igralec␣je␣")

desni_stzmag = 0
levi_stzmag = 0

stRund = 5

while stRund:
        sleep(uniform(5,10))
        led.on()
        if leva_tipka.is_pressed:
                print(levo_ime + "␣je␣zmagal")
                levi_stzmag += 1
        else:
                print(desno_ime + "␣je␣zmagal")
                desni_stzmag += 1
```

```
        print(levi_stzmag + ":" + desni_stzmag)

        stRund -= 1
```

Sedaj pa jo preizkusi.

Hitri prsti

Cilji:

- boljše razumevanje funkcije *print*()
- ponovitev funkcij modula *Button*
- boljše razumevanje zanke *while* in stavka *if*

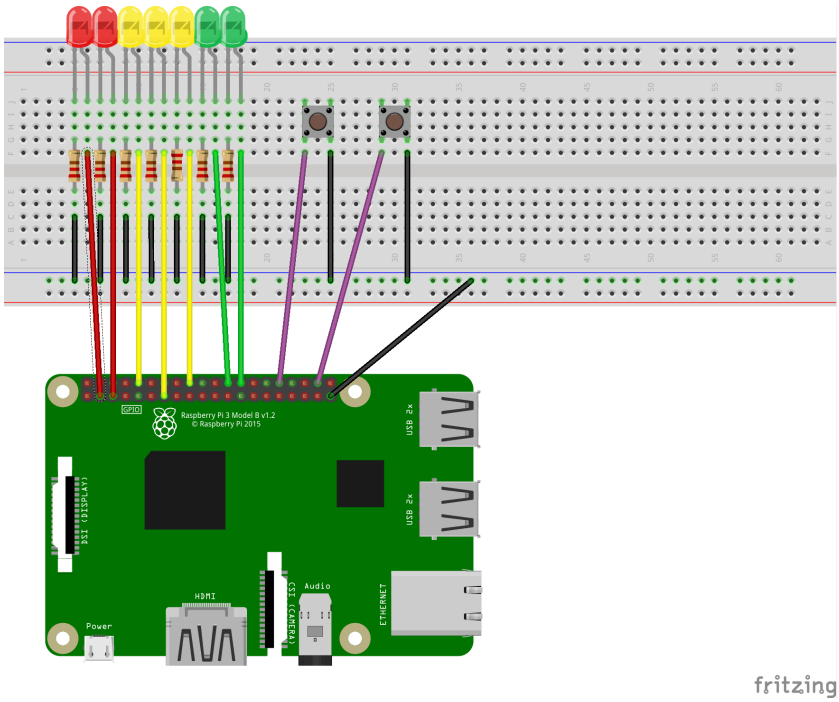
Prepričaj enega od družinskih članov, naj se pomeri s tabo v igri hitrih prstov.

Pravila: Naenkrat igrata oba igralca. Igra poteka v več krogih (rundah). Vsak krog predstavlja en val, v katerem se diode prižigajo in ugašajo od prve do zadnje. Igralec s pritiskom na tipko dobi število točk, ki jih prinaša dioda, ki je svetila v trenutku, ko je bila tipka pritisnjena. Prva dioda prinese 1 točko, druga 2 itd. Če hočeš veliko število točk, moraš biti spreten in hiter. Cilj naloge je torej pritisniti tipko ravno takrat, ko sveti dioda, ki prinese največ točk. Diode se prižigajo krožno, kar pomeni, da za sedmo diodo zasveti prva. Po vsakem krogu naj se izpiše število točk, ki jih je tekmovalec takrat pridobil. Sestavi vezje z dvema tipkama in 7 diodami, tako kot kaže slika 4.8.

Potek:

1. Ustvarimo novo datoteko ter jo shranimo kot *hitri_prsti.py*. Vključimo potrebne knjižnice in module ter 7 diod in dve tipki. Pri tem pazimo, da so številke v oklepajih enake povezanim nožicam. Vrstni red prižiganja diod bo pomemben, zato bodi pozoren.

```
from gpiozero import LED, Button
from time import sleep
```



Slika 4.8: Vezje s 7 diodami in dvema tipkama

```
tipka1 = Button(12)
tipka2 = Button(20)
led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)
```

2. Sedaj pa sledi drugi del. Osnova naloge je krožno valovanje. Diode se prižigajo in ugašajo ena za drugo, ko ugasne zadnja (sedma) LED dioda, pa se ponovno prižge prva itd. Diode zapišemo v seznam, ki ga bomo uporabljali pri prižiganju in ugašanju diod. Uporaba seznama nam omogoča manj pisanja ter s tem lepši pregled nad kodo. Prižiganje in ugašanje se bo torej dogajalo znotraj zanke *while*, zato zapišemo:

```
seznamLedic = [led1, led2, led3, led4, led5, led6, led7]

while True:
    for led in seznamLedic:
```

```
        led.on()
        sleep(1)
        led.off()
```

Če kodo shranimo in prevedemo, bomo videli, da je naše valovanje precej počasno. Zdi se lahko že skoraj dolgočasno. Zato bomo v namene igrice čas v oklepaju z 1 sekunde skrajšali na 0,05 sekunde.

3. Če želimo, da se zanka izvede točno tolikokrat, kot smo si zamislili, bomo potrebovali spremenljivko *stRund*. Ta se bo vsakič, ko se bo zanka izvedla, zmanjšala za 1. Namesto vedno izpolnjenega pogoja $True$ pa bomo uporabili pogoj $stRund > 0$. Ker spremenljivko vsak krog zmanjšamo za ena, bo njena vrednost čez nekaj krogov enaka 0. Takrat pogoj ne bo več izpolnjen in zanka *while* se bo zaključila.

```
stRund = 5

while stRund > 0:
    for led in seznamLedic:
        led.on()
        sleep(1)
        led.off()
        stRund -= 1
```

4. Ker želimo točke posamezne diode, uporabimo novo spremenljivko $stTock$. Ta bo vsakič, ko se bo prižgala nova dioda, večja. Ker pa želimo, da je v vsakem krogu prva dioda vredna 1 točko, ustvarimo spremenljivko znotraj zanke *while*. Spremenljivke, ki jo ustvarimo znotraj zanke, zunaj nje ne moremo uporabljati. Spremenljivka $stTock$ bo torej pred vsakim začetkom vala enaka 0 ter pri vsaki naslednji diodi večja za ena.

```
while stRund > 0:
        stTock = 0
    for led in seznamLedic:
        stTock +=1
        led.on()
        sleep(1)
        led.off()
        stRund -= 1
```

5. Vse, kar potrebujemo sedaj, pa je preverjanje stanja tipk. Če je pritisnjena $tipka1$ v času, ko je svetila šesta dioda, naj se izpiše: *'prvi: 6'* oz.

'drugi: 6', če je bila pritisnjen *tipka*2 v času, ko je svetila šesta dioda.

```
if tipka1.is_pressed():
        print("prvi:␣" + stTock)
if tipka2.is_pressed():
        print("drugi:␣" + stTock)
```

Naš končni program izgleda tako:

```
from gpiozero import LED, Button
from time import sleep

tipka1 = Button(12)
tipka2 = Button(20)
led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

seznamLedic = [led1,led2,led3,led4,led5,led6,led7]

stRund = 5

while stRund > 0:
    stTock = 0
    for led in seznamLedic:
        stTock +=1
        led.on()
        sleep(0.05)
        if tipka1.is_pressed:
            print("prvi:" + stTock)
        if tipka2.is_pressed:
            print("drugi:" + stTock)
        led.off()

    stRund -= 1
```

Dodatne naloge:

1. Dodaj spremenljivki $vsotaTock1$ ter $vsotaTock2$, ki predstavljata skupno število točk vsakega posameznega igralca. Razmisli, kje in za koliko vsak krog povečaš spremenljivki. Ko se igra zaključi, izpiši vrednosti obeh spremenljivk.

2. Število pritiskov na val je pomembno in ker igra ne sme dopuščati goljufanja, vključi spremenljivk *stPritiskov*1 ter *stPritiskov*2 in razmisli, kolikšno vrednost imata pred vsakih valom, kje ju spreminjamo ter kje preverjamo.

   V stavku *if* lahko preverjamo dve stvari, če med njiju napišemo besedo *and*. Stavek *if* se izvede, če sta izpolnjena tako prvi kot drugi pogoj.

   Primer:

   ```
   if 2>1 and 1==1:
           print("Moj␣Raspberry␣Pi")
   ```

   Veljata oba pogoja, zato se izpiše stavek *'Moj Raspberry Pi'*.

   Dve vrednosti med sabo primerjamo z dvema enačajema (==).

Rešitve: 1.

```
from gpiozero import LED, Button
from time import sleep

tipka1 = Button(12)
tipka2 = Button(20)
led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

seznamLedic = [led1,led2,led3,led4,led5,led6,led7]

stRund = 5
vsotaTock1 = 0
vsotaTock2 = 0

while stRund > 0:
    stTock = 1
    stPritiskov1 = 0
    stPritiskov2 = 0
    for led in seznamLedic:
        led.on()
        sleep(0.05)
```

```
        if tipka1.is_pressed:
            vsotaTock1 += stTock
            print("prvi:" + stTock)
        if tipka2.is_pressed:
            vsotaTock2 += stTock
            print("drugi:" + stTock)
        led.off()
        stTock += 1

    stRund -= 1

print(vsotaTock1)
print(vsotaTock2)
```

2.

```
from gpiozero import LED, Button
from time import sleep

tipka1 = Button(12)
tipka2 = Button(20)
led1 = LED(2)
led2 = LED(3)
led3 = LED(15)
led4 = LED(27)
led5 = LED(24)
led6 = LED(8)
led7 = LED(7)

seznamLedic = [led1,led2,led3,led4,led5,led6,led7]

stRund = 5
vsotaTock1 = 0
vsotaTock2 = 0

while stRund > 0:
    stTock = 1
    stPritiskov1 = 0
    stPritiskov2 = 0
    for led in seznamLedic:
        led.on()
        sleep(0.05)
        if tipka1.is_pressed and stPritiskov1 == 0:
            vsotaTock1 += stTock
            print("prvi:" + stTock)
            stPritiskov += 1
        if tipka2.is_pressed and stPritiskov2 == 0:
            vsotaTock2 += stTock
            print("drugi:" + stTock)
            stPritiskov += 1
```

```
        led.off()
        stTock += 1

    stRund -=1

print(vsotaTock1)
print(vsotaTock2)
```

Pomeri se v igri.

Nabor funkcij za delo z diodo in tipko	
LED	
$from\ gpiozero\ import\ LED$	vključitev modula za diode
$on()$	prižge diodo
$off()$	ugasne diodo
$blink()$	omogoča utripanje diode, dokler programa ne prekinemo
$toggle()$	funkcija, ki spremeni stanje diode; če sveti, jo ugasne, če je ugasnjena, jo prižge
Button	
$from\ gpiozero\ import\ Button$	vključitev modula za tipko
$wait\_for\_press()$	funkcija, ki čaka, da je tipka pritisnjena; ko je pritisnjena, se izvedejo ukazi, napisani za njo
$is\_pressed()$	funkcija, ki preverja, če je tipka pritisnjena; največkrat se uporabi v kombinaciji z ukazom if; če je pritisnjena, bo pogoj izpolnjen, če ni, bo pogoj neizpolnjen.
$when\_pressed()$	največkrat se uporabi v kombinaciji z drugimi funkcijami, za katere želimo, da se izvedejo vsakič, ko bo tipka pritisnjena

Nabor ukazov v Pythonu	
$\#$	se uporablja, če želimo napisati opazke zase in ne želimo, da jih računalnik prebere
def	omogoča definiranje funkcije
$for()$	zanko for se običajno uporablja, kadar želimo del kode ponoviti tolikokrat, kot smo si zamislili
if	če je pogoj resničen, se nekaj zgodi
$if...else$	če je pogoj resničen, se zgodi en nabor stvari, če pogoj ni resničen, se zgodi drug nabor stvari
$import$	omogoča vključitev modula ali funkcije
$input()$	funkcija, ki zahteva vnos niza (Stringa)
$vreca = [$"$jabolko$","$hruska$", "$banana$","$pomaranca$"$]$	primer definicije lista v Pythonu; list lahko vsebuje nize (Stringe), ki so ločeni z vejico in zapisani v oglatem oklepaju
$ime = vrednost$	primer spremenljivke
$print()$	funkcija, ki izpiše vsebino oklepaja
$random$	modul, ki omogoča vključitev funkcij, povezanih z naključnim izbiranjem; primer funkcije $uniform()$
$time$	modul, ki omogoča različne funkcije v povezavi s časom, kot na primer $sleep()$
$while$	če je pogoj resničen, se zanka $while$ stalno ponavlja

4.2 Minecraft

Minecraft je popularna igra za grajenje svetov. Poleg tega, da jo najdeš na svojem Raspberryju, pa lahko ukaze pišeš tudi v Pythonu.

4.2.1 Začetki

Da program zaženeš, moraš v terminal ($Ctrl + Alt + T$) najprej vpisati $minecraft - pi$ ter pritisniti enter. Lahko pa ga poiščeš med igrami po kliku na Meni.

Ko se naloži, klikni na gumb Start Game ter nato na Create New. Odpre se ti Minecraftov svet, po katerem se premikaš tako:

tipka	akcija
W	naprej
A	levo
S	nazaj
D	desno
E	arhiv
Space	skok
dvojni Space	leti/padi
ESC	pavza/glavni meni
Tab	sprosti miško
leva tipka na miški	postavi blok
desna tipka na miški	uniči blok v bližini

Predmet lahko izbereš na več načinov. Po pritisku tipke E se ti odpre arhiv, kot kaže slika 4.9, po katerem se lahko premikaš s tipkami na tipkovnici. Predmet izbereš s pritiskom na Enter ali pa nanj enostavno klikneš. Izbran predmet v roki postaviš pred sabe z levo tipko na miški, z desno pa ga lahko

odstraniš. Da pa nam pri menjavanju blokov ni treba vedno odpreti arhiva, imamo na zaslonu pripravljen nabor zadnjih uporabljenih blokov. Med njimi se sprehajamo z drsnikom na miški. Če izbereš meč, lahko odstranjuješ bloke pred sabo ali pa izkoplješ luknjo.

Slika 4.9: Arhiv

Preizkusi še tipko presledek. Če ga pritisneš dvakrat zaporedoma, boš poletel v zrak. Če ga nato pritiskaš le enkrat, boš z vsakim pritiskom višje v zraku. Ko se ti zazdi, da si dovolj visoko, dvakrat pritisni presledek in padel boš nazaj na tla.

Poglejmo si še, kako lahko Minecraft nadzorujemo s pomočjo Pythona. Če se še vedno nahajamo v Minecraftu, miškin fokus zunaj njega pridobimo s klikom na tabulator (tipka Tab). V meniju poiščemo in odpremo *Python 2 (IDLE)* ter okni pomaknemo eno zraven drugega. S klikom na File, New File ustvarimo novo datoteko.

Če se hočemo povezati z igro, moramo vpeljati knjižnico za delo z Minecraftom. Za začetek pa hočemo na zaslon izpisati tudi sporočilo *'Pozdravljen svet'*:

```
from mcpi.minecraft import Minecraft

mc = Minecraft.create()
```

```
mc.postToChat("Pozdravljen svet!")
```

Datoteko shrani in poženi z F5. Ko se tvoja koda zažene, bi moral opaziti sporočilo, ki ga prikazuje slika 4.10.

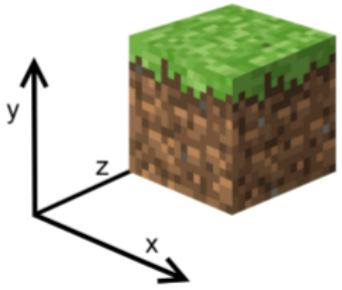
Slika 4.10: Pozdravljen svet v Minecraftu

4.2.2 Lokacija

Igra je približek realnega sveta. Za to uporabljajo koordinate na sliki 4.11. Te nam povedo, kje v prostoru se nahajamo.

Če hočemo v Pythonu izvedeti svojo lokacijo znotraj igre, zapišemo:

```
pos = mc.player.getPos()
```



Slika 4.11: Koordinate

Spremenljivka *pos* sedaj vsebuje našo lokacijo. Do vsake informacije posebej dostopam na sledeč način:

```
x = pos.x
y = pos.y
z = pos.z
```

Sedaj x, y in z vsebujejo števila, ki nam povedo, kje natančno se nahajamo.

Zapomni si, da funkcija *getPos*() pove trenutno lokacijo, če pa se medtem premaknemo, pa moramo funkcijo poklicati ponovno.

Dodatno: Če lahko tako enostavno pridobimo svojo lokacijo, jo je tako enostavno tudi spremeniti, da bi se torej brez pritiskanja tipke presledek lahko dvignili v zrak?

Tako kot lahko izvemo svojo trenutno pozicijo, pa jo tudi nastavimo. Na ta način se lahko preteleportiramo. Zato uporabimo funkcijo *setPos*().

```
pos = mc.player.getPos()
x = pos.x
y = pos.y
z = pos.z

mc.player.setPos(x, y+100, z)
```

Ker smo y, ki nam pove, kako visoko smo, povečali, se kar naenkrat znajdemo v zraku. In ker nimamo podlage, na kateri bi stali, začnemo padati. Na ta način se lahko prestavimo tudi nižje.

```
mc.player.setPos(x, y-3, z)
```

Če y zmanjšamo, se znajdemo pod zemljo, peskom ali v skali, odvisno, na kakšni podlagi stojimo. Iz luknje, v kateri se znajdemo, se lahko tudi izkopljemo. Razmisli, kateri pripomoček lahko uporabiš, da prideš do zraka.

> Namig: Uporabi poseben pripomoček za razbijanje blokov.

4.2.3 Vse o blokih

S funkcijo *setBlock*() lahko postavimo blok na določeno lokacijo.

```
pos = mc.player.getPos()
x = pos.x
y = pos.y
z = pos.z

mc.setBlock(x+1, y, z, 1)
```

Nekje v tvoji bližini se je pojavil siv blok. Če ga ne vidiš takoj, se postavi v Minecraft in se obračaj na istem mestu, dokler ga ne vidiš, tako kot kaže slika 4.12.

Slika 4.12: Uporaba funkcije *setBlock*() s kamnom

V oklepaj pri funkciji *setBlock*() napišemo najprej vrednosti x, y in z, nato pa sledi številka bloka. Vrednosti x, y in z so povezane z igralcem; ker zapišemo $x + 1$, postavimo blok 1 za eno mesto stran od igralca. Številka bloka pa je tip bloka. 1 je kamen. Poskusiš lahko še z drugimi tipi bloka. 0 je zrak, 2 je trava, zemlja pa je 3. Ostale bloke in njihove številke si lahko pogledaš na spletni strani `http://minecraft-ids.grahamedgecombe.com/`.

Poleg tebe se bo pojavila roža, tako kot kaže slika 4.13, če zapišeš:

```
mc.setBlock(x+1, y, z, 38)
```

Obstaja tudi funkcija, ki nam olajša delo pri postavljanju večjega števila blokov. Če hočemo na primer zgraditi zid in ne želimo vsakega bloka postaviti ročno, zapišemo:

```
pos = mc.player.getPos()
```

Slika 4.13: Uporaba funkcije *setBlock*() z rožo

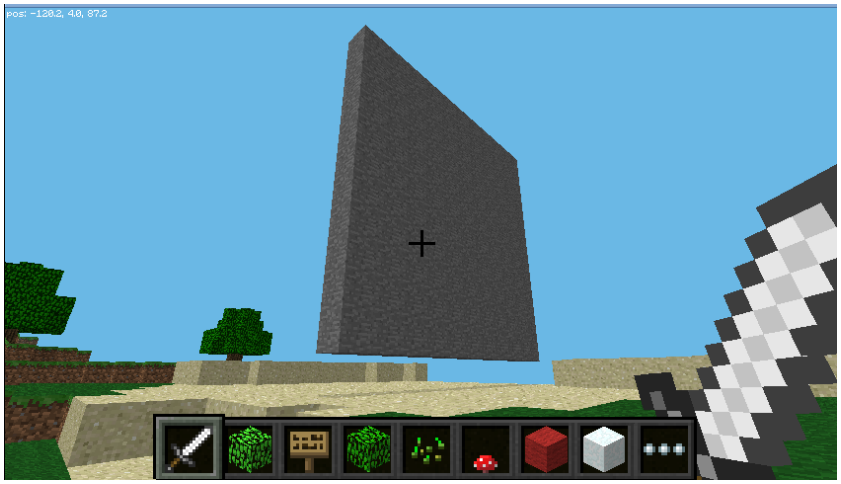
```
x = pos.x
y = pos.y
z = pos.z

kamen = 1

mc.setBlocks(x+1, y+1, z+1, x+1, y+11, z+11, kamen)
```

To bo ustvarilo zid v naši bližini. Če se od njega oddaljimo, bo zid izgledal tako, kot kaže slika 4.14. Funkciji podamo začetne vrednosti ($x + 1$, $y + 1$, $z + 1$) ter končne vrednosti ($x + 1$, $y + 11$, $z + 11$). Zid se bo v daljino raztezal med vrednostma $x + 1$ in $x + 1$, kar pomeni, da bo zid dolg 1 blok. V višino se bo raztezal med vrednostma $y + 1$ in $y + 11$, kar pomeni, da bo visok 10 blokov. V širino pa se bo raztezal med vrednostma $z + 1$ in $z + 11$, kar pomeni, da bo širok 10 blokov.

4.2.4 Puščanje sledi

Pogledali si bomo, kako lahko za sabo puščamo bloke. Pri tem si bomo pomagali s trenutnimi vrednostmi v prostoru. Namen je, da vsakič preverimo, kje smo, ter na istem mestu pustimo blok. Za sabo lahko puščamo poljubne bloke, mi se bomo odločili za rožo. Zato bomo zapisali:

```
from mcpi.minecraft import Minecraft
```



Slika 4.14: Uporaba funkcije *setBlocks*()

```
from time import sleep

mc = Minecraft.create()

while True:
        pos = mc.player.getPos()
        x = pos.x
        y = pos.y
        z = pos.z

        mc.setBlock(x,y,z,38)
        sleep(0.1)
```

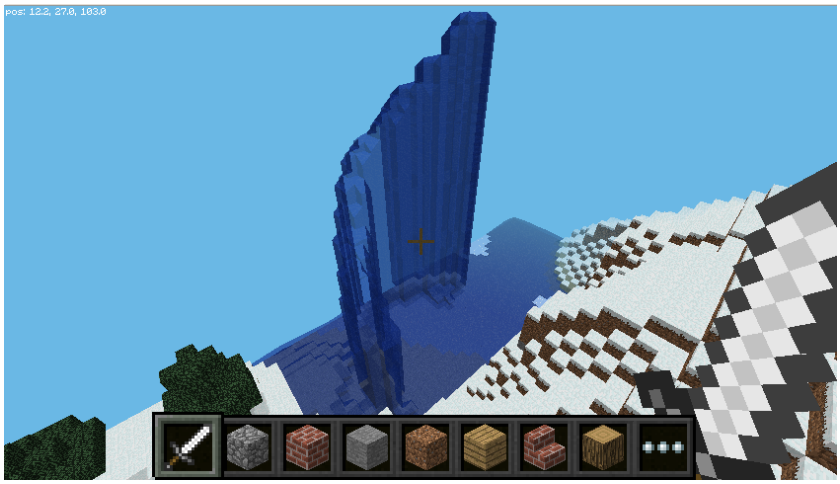
Ker se bomo premikali, se bodo spreminjale tudi vrednosti x, y, z. Zato uporabimo zanko *while*, ki stalno pridobiva našo lokacijo.

Če se sedaj nekaj časa premikamo in se nato obrnemo nazaj, bomo za sabo videli rože, ki smo jih pustili za sabo. Če pa se dvignemo od tal in nekaj časa letimo po nebu, se bo za nami ustvarila sled.

Program ustavimo v Pythonovem oknu s pritiskom na tipki *Ctrl + C*.

Če namesto rož uporabimo vodo, lahko ustvarimo nekaj podobnega, kot kaže slika 4.15.

Preizkusi sam.

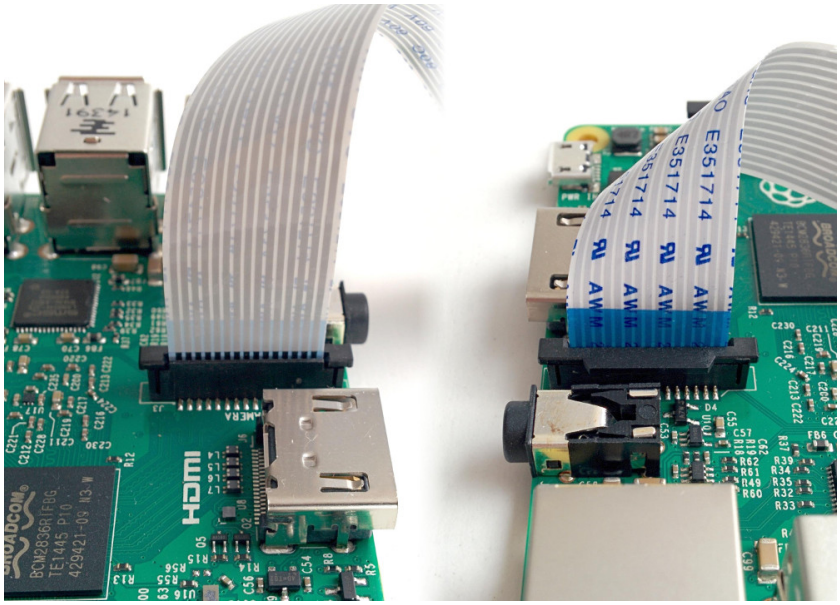


Slika 4.15: Puščanje sledi z vodo

Nabor ukazov za igro Minecraft	
$from\ mcpi.minecraft\ import\ Minecraft$	omogoča vključitev modula za delo v Minecraftu
$mc = Minecraft.create()$	ustvari Minecraftov objekt, s katerim omogoči povezavo z igro Minecraft Pi
$postToChat()$	deluje podobno kot funkcija $print()$, le da ta besedilo v oklepaju izpiše na Minecraftov zaslon
$player.getPos()$	omogoča pridobitev vrednosti x, y in z naše lokacije
$player.setPos(x, y, z)$	funkcija omogoča nastavitev pozicije igralca; vrednosti x, y in z zapišemo v oklepaj
$setBlock(x, y, z, t)$	funkcija postavi blok tipa t na mesto z vrednostmi x, y, z
$setBlocks(x, y, z, xx, yy, zz, t)$	funkcija, ki omogoča hkratno postavitev več blokov tipa t med dva nabora vrednosti

4.3 Raspberry Pi in Minecraft

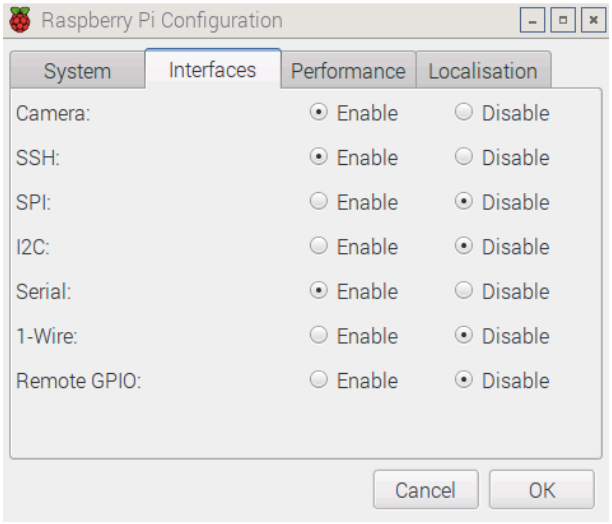
Za vse tri spodnje naloge bomo potrebovali Raspberryjevo kamero (Picamero). Picamero je enostavno uporabljati. Za začetek moramo vključiti kamero v Raspberry, kar moramo storiti, še preden ga vključimo v elektriko. Najprej moramo ugotoviti, kam priključiti kamero. Ko najdemo priključek, z dvema prstoma dvignemo pokrovček. Nato kamero vanj vstavimo tako, kot kaže slika 4.16.



Slika 4.16: Pravilna namestitev kamere

Ko je kamera pravilno nameščena, zapremo pokrovček in pazimo, da je medtem ne premaknemo. Priključimo Raspberry Pi ter pod Menijem med Preferences poiščemo Raspberry Pi Configuration.

Prepričamo se, da je kamera omogočena. Če ni, kliknemo gumb Enabled, tako kot kaže slika 4.3. Odpre se okno, na katerega s klikom na gumb Yes potrdimo, da želimo resetirati Raspberry Pi.



Slika 4.17: Omogočimo uporabo kamere

Sedaj je kamera pripravljena za uporabo. Knjižnica *picamera* nam omogoča njeno uporabo znotraj programskega jezika Python.

```
from picamera import PiCamera

kamera = PiCamera()
```

Funkcije, ki jih knjižnica omogoča, so: *start_preview*(), *stop_preview*, *capture*(), *start_recording*(), *stop_recording*() ...

Sedaj pa je čas, da jih vsaj nekaj uporabimo v naslednjih treh nalogah.

4.3.1 S tipko do slike

Kamera naj ob vsakem pritisku na tipko posname fotografijo. Slike shranjuj v skupno mapo, vsaka od slik naj bo poimenovana generično.

Potek:

1. Pravilno priključimo kamero.
2. Odpremo terminal (*Ctrl* + *Alt* + *T*), v katerega vpišemo:

```
raspistill -k
```

Na zaslonu se prikaže predogled. Naj te ne skrbi, če je slika narobe obrnjena. To lahko popravimo kasneje. Pritisnemo *Ctrl* + *C*, da izklopimo

predogled.

3. Odpri *Python 2 (IDLE)*, odpri novo datoteko ter vanjo zapiši:

```
from picamera import PiCamera
from time import sleep

kamera = PiCamera()

kamera.start_preview()
sleep(3)
kamera.capture('/home/pi/Desktop/slika1.jpg')
kamera.stop_preview()
```

Funkcija *start_preview* odpre predogled in je nujen korak pred slikanjem. Če želimo zajeti sliko, uporabimo funkcijo *capture*(). Poimenuj datoteko *slike.py* ter jo shrani na poljubno mesto.

S klikom na F5 zaženi program. Preveri, kam kaže tvoja kamera, ali pa odpri datoteko *image.jpg* na namizju. Vanjo se je shranila slika, ki smo jo ustvarili s:

```
kamera.capture()
```

Če je slika obrnjena narobe, pred vrstico

```
kamera.start_preview()
```

dodaj:

```
kamera.rotation = 180
```

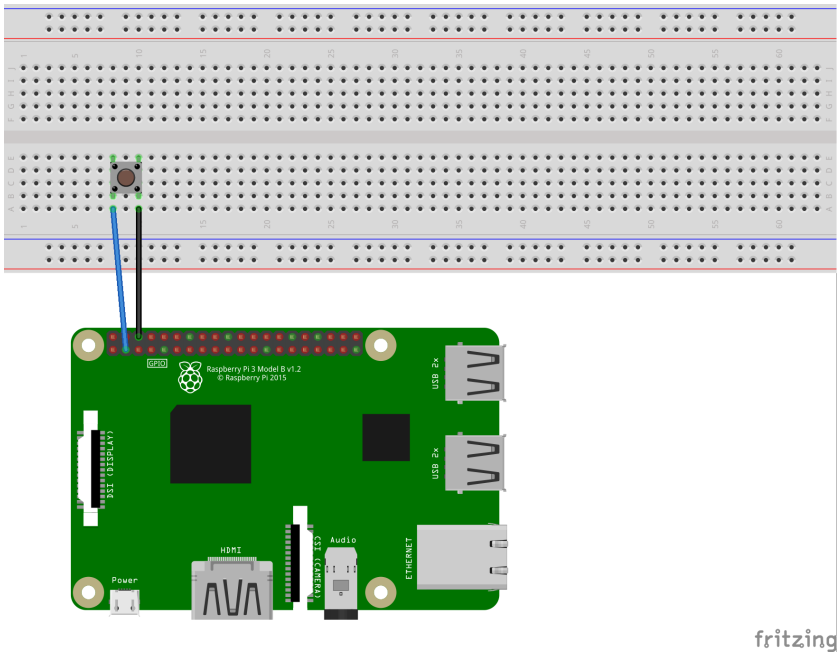
Če datoteko ponovno zaženeš, se bo ustvarila nova slika, ki bo nadomestila prejšnjo. Tokrat bo slika pravilno obrnjena. Če vrstice ne potrebuješ, jo lahko izbrišeš, v nasprotnem primeru naj ostane del kode.

4. Vezje, ki ga bomo potrebovali pr tej nalogi, kaže slika 4.18.

Tipka naj bo torej z eno stranjo povezana na GND, z drugo pa na poljubno nožico. Številko nožice bomo potrebovali pri pisanju kode, kjer bomo najprej vključili knjižnico za delo s tipkami:

```
from gpiozero import Button

tipka = Button(2)
```

Kasneje pa bomo nadomestili funkcijo *sleep*() za funkcijo, ki čaka na pritisk tipke. Koda, ki jo shranimo in prevedemo, izgleda tako:



Slika 4.18: Vezje z eno tipko

```
from camera import PiCamera
from time import sleep
from gpiozero import Button

tipka = Button(2)
kamera = PiCamera()

kamera.start_preview()
tipka.wait_for_press()
kamera.capture('/home/pi/Desktop/slika2.jpg')
kamera.stop_preview()
```

Kamera slike ne bo zajela, dokler ne pritisnemo na tipko, ki smo jo povezali z Raspberryjem. Ko je tipka pritisnjena, se na namizju ustvari slika z imenom *slika2.jpg*.

5. Sliko shranimo v posebno mapo. Mapo ustvarimo v terminalu tako:

```
mkdir slike
```

To bo ustvarilo mapo z imenom *slike*. V mapo bomo shranili več slik, zato moramo temu primerno popraviti kodo. Ob pritisku na tipko se program ustavi. Če dodamo zanko, bomo naredili več slik, pri tem pa program zagnali

le enkrat.

```
okvir = 1

def slikaj():
        kamera.start_preview()
        kamera.capture('/home/pi/slike/okvir%03d.jpg' %okvir)
        okvir +=1
        kamera.stop_preview()

tipka.when_pressed = slikaj
```

Kodo smo nekoliko spremenili. Ob vsakem pritisku se izvede funkcija *slikaj*(). Ta vključi predogled, vzame sliko, pri tem poveča spremenljivko *okvir* ter predogled zaključi.

Uporaba spremenljivke *okvir* omogoča, da se bodo slike shranjevale kot imena trimestnih številk, začenši z 001. Imena slik bodo nato vsakič za eno večja (002, 003 itn.). Na ta način vemo, kako si sledijo po vrstnem redu.

4.3.2 Foto kotiček V minecraftu

Prva stvar, ki jo moramo storiti, je med aplikacijami poiskati Minecraft. Ko kliknemo nanj, ustvarimo nov svet ali pa odpremo že obstoječega. Prenesemo okno Minecrafta na eno stran zaslona, pritisnemo tabulator (tipka Tab) ter z miško kliknemo na Meni. Odpremo *Python 2 (IDLE)*, ki zažene Pythonov vmesnik. Odpremo novo datoteko, jo poimenujemo in shranimo. Nato pa vanjo vtipkamo:

```
from mcpi.minecraft import Minecraft
from picamera import PiCamera
from time import sleep

mc = Minecraft.create()
camera = PiCamera()

mc.postToChat("Pozdravljen␣svet!")
```

Shrani in zaženi z F5 ter preveri, kaj se zgodi.

Da preverimo, ali kamera deluje, bomo dodali nekaj vrstic kode:

```
camera.start_preview
sleep(2)
camera.capture('/home/pi/Desktop/selfi.jpg')
```

```
camera.stop_preview()
```

Pred slikanjem se za dve sekundi pokaže predogled, v katerem lahko popravimo pozo ter se nasmehnemo. Slika se shrani v datoteko, imenovano *selfi.jpg* na namizje.

Če si pripravljen, shrani in zaženi program.

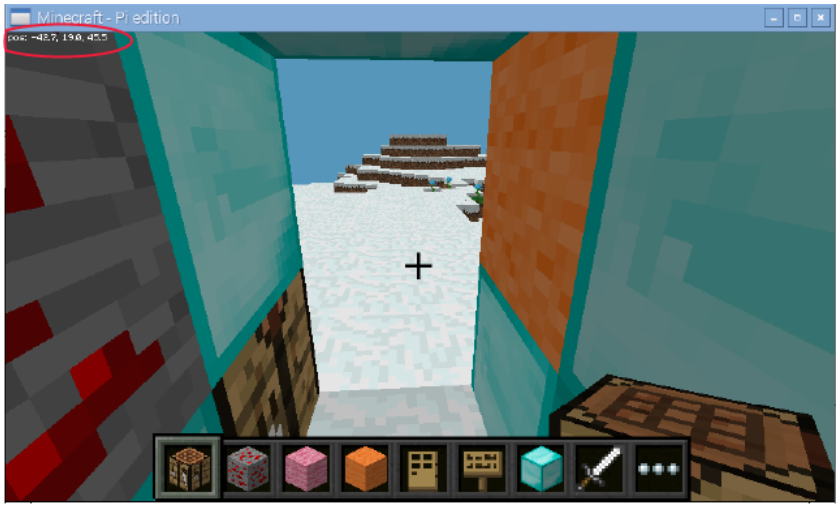
Sedaj pa potrebujemo hišico. Poišči primeren prostor ter začni graditi. Pri gradnji se lahko poslužuješ kateregakoli bloka. Hiška je lahko poljubne velikosti in oblike. Pogoj je le, da je znotraj hiške vsaj en prazen prostor, da igralec lahko vstopi.

Slika 4.19: Hiška v svetu Minecrafta

Ko hiško zgradiš, bo izgledala nekako tako kot hiška na sliki 4.19. Preveri, če lahko igralec gre v njo ter se postavi na sprožilno mesto. Tako se imenuje zato, ker sproži funkcijo, ki slika. Ko se premaknemo v hiško, je sprožilno mesto ravno mesto, na katerem se trenutno nahajamo, zato bomo za pridobitev vrednosti uporabili funkcijo $player.getPos()$. Vse tri vrednosti bodo zapisane v levem zgornjem kotu. Vrednosti si zapišemo, ker jih bomo potrebovali.

Ker želimo, da se nam izpiše, kdaj bomo v hišici, zapišemo:

```
while True:
```



Slika 4.20: Koordinate sprožilnega mesta

```
        pos = mc.player.getPos()
        x = pos.x
        y = pos.y
        z = pos.z
        sleep(3)
        if x == -42.7 and y == 19.0 and z == 45.5:
                mc.postToChat("Si␣v␣hiski!")
```

Tvoja lokacija bo drugačna, to samo pomeni, da si hiško zgradil na drugem mestu. Če hočeš, da se izpiše *'Si v hiški!'*, uporabi svoje vrednosti x, y, z. Slika 4.20 prikazuje notranjost hiške. V levem kotu pa najdeš svoje vrednosti x, y, z.

Zanka *while* preverja našo lokacijo ter dokler ne pridemo v hiško, ne izpiše ničesar.

Shrani ter zaženi kodo. Postavi se v hiško ter poglej, če se je izpisalo sporočilo.

Sedaj, ko imamo vse potrebne dele kode, jih moramo le še združiti. Preden se sproži kamera, bomo dodali še opomnik.

Končna verzija izgleda tako:

```
from mcpi.minecraft import Minecraft
from picamera import PiCamera
from time import sleep

mc = Minecraft.create()
```

```
camera = PiCamera()

while True:
        pos = mc.player.getPos()
        x = pos.x
        y = pos.y
        z = pos.z

        if x == -42.7 and y == 19.0 and z == 45.5:
                mc.postToChat("Si␣v␣hiski!")
                sleep(1)
                mc.postToChat("Nasmehni␣se!")
                sleep(1)
                camera.start_preview()
                sleep(2)
                camera.capture('/home/pi/Desktop/selfi.jpg')
                camera.stop_preview()
sleep(3)
```

Če se ob vstopu v hiško ne izpiše nič, se premikaj toliko časa, da bodo vrednosti x, y in z res prave. Če boš stal na istem mestu, te bo slikalo večkrat. Vendar pa bo slika le ena, ker imena slike ne spreminjamo. Razmisli in spremeni kodo na tak način, da bo ime slike vsakič drugačno.

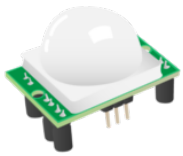
4.3.3 Detektor staršev

Če želiš izvedeti, kdo ti brska po sobi, medtem ko te ni doma, ti je naslednja naloga pisana na kožo. Senzor za premikanje bo sprožil kamero, ki bo posnela vsiljivca v tvoji sobi.

Nekaj o infrardečem senzorju, ki zaznava premikanje

Pri tej nalogi si bomo pomagali s pasivnim infrardečim senzorjem (senzorjem PIR na sliki 4.21), ki zaznava premikanje. Take senzorje se uporablja pri zaznavanju vlomilcev in so navadno nameščeni v zgornjem kotu sobe. Senzor zaznava 'tople' objekte, ki oddajajo infrardeče valove. Ker mi teh valov nismo sposobni zaznati, uporabljamo naprave, ki jih lahko.

Senzor torej zazna katerikoli premikajoč objekt v sobi. Z notranjim delovanjem senzorja se ne bomo preveč obremenjevali, si pa bomo pogledali njegovo zunanjost. Vsebuje namreč 3 nožice, preko katerih ga bomo povezali



Slika 4.21: PIR senzor

z Raspberryjem.

Potek:

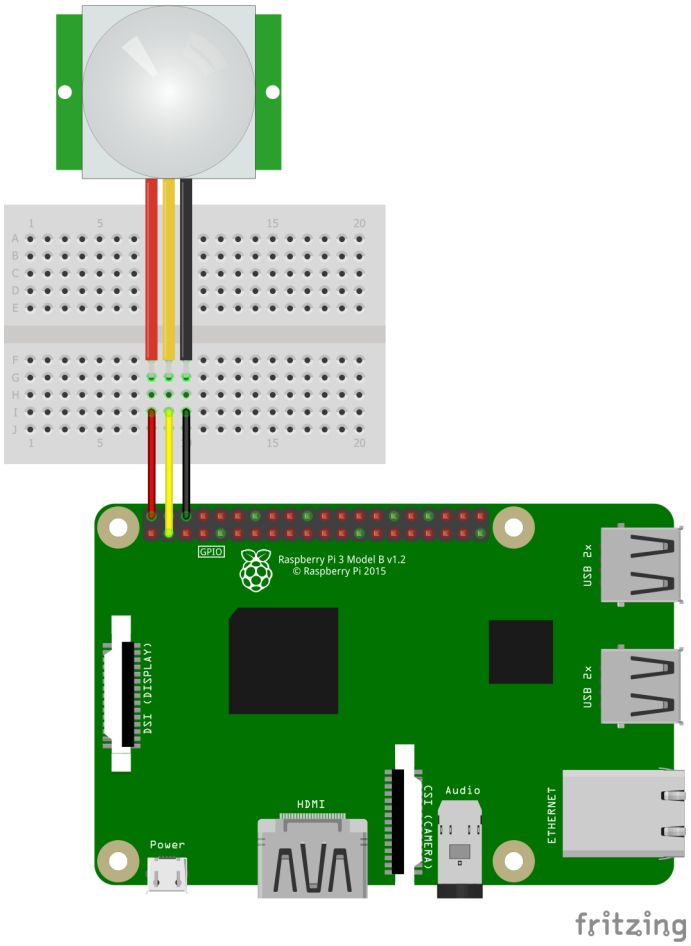
1. Preden prižgemo Raspberry Pi, nanj povežemo infrardeč senzor, tako kot kaže slika 4.22, ter priključimo kamero.

Pri povezava se prepričaj, da bo

- nožica VCC na senzorju povezana na nožico +5V na Raspberryju (nožica VCC dovaja energijo senzorju)
- nožica GND povezana z nožico GND na Raspberryju (to zaključi krog)
- nožica OUT povezana z nožico GPIO 4 na Raspberryju (preko nožice 4 senzor sporoči Raspberryju, da je zaznal premikanje)

2. Raspberry Pi sedaj povežemo z elektriko. Nato odpremo *Python 2 (IDLE)*, ustvarimo novo datoteko ter začnemo pisati. Najprej bomo vključili knjižnico, potrebno za delo s senzorjem. To je knjižnica *gpiozero*, ki smo jo uporabljali tudi za diode in tipke. Zapišemo, na kateri nožici se nahaja naš senzor, ter v neskončni zanki preverjamo, če je senzor zaznal premikanje. Če je pogoj izpolnjen, izpišemo *'Zaznal sem premikanje!'*. Kasneje bomo ta del primerno spremenili.

```
from gpiozero import MotionSensor

senzor = MotionSensor(4)

while True:
        if senzor.motion_detected:
                print("Zaznal␣sem␣premikanje!")
```

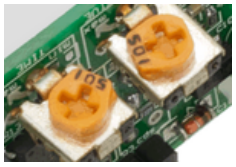


Slika 4.22: Vezava senzorja

Program ugasnemo ročno s pritiskom na *Ctrl + C*. Shrani kodo, nato pa jo zaženi s pritiskom na tipko F5. Vsakič ko senzor zazna gibanje, se bo izpisalo *'Zaznal sem premikanje!'*.

Na senzorju bi moral opaziti tudi dva oranžna kroga, ki ju lahko premikaš z izvijačem. Imenovana sta potenciometra in omogočata fizično nastavitev občutljivosti senzorja ter časa zaznavanja premikanja. Nastavi občutljivost senzorja na maksimum, čas zaznavanja pa na minimum, tako kot kaže slika 4.23. Kasneje ju lahko tudi spremeniš.

3. Vključimo knjižnico, ki omogoča uporabo vnaprej pripravljenih funkcij za delo s kamero. V zanki *while* uporabimo funkcijo senzorja, ki nam



Slika 4.23: Potenciometra

omogoča čakanje na premik. Ko naš senzor zazna premikanje, pričnemo snemati, nato pa naj senzor počaka, da se premikanje zaključi. Za tem se zaključi tudi snemanje.

```
from gpiozero import MotionSensor
from picamera import PiCamera

kamera = PiCamera()
senzor = MotionSensor(2)

while True:
        senzor.wait_for_motion()
        #imeposnetka
        kamera.start_recording(imeposnetka)
        senzor.wait_for_no_motion()
        kamera.stop_recording()
```

Ker želimo video posneti vsakič, ko vsiljivec stopi v sobo, potrebujemo zato ob vsakem vstopu drugačno ime datoteke, drugače bo vsak nov posnetek povozil prejšnjega. Zato bomo vsak nov posnetek poimenovali avtomatsko in ne ročno. Najlažji način za to je uporaba datuma in časa, saj ta nikoli več ne bosta ista.

Za uporabo datuma in časa bomo potrebovali knjižnico *datetime*. Če bi bil danes 15. 9. 2016, čas pa 10:24:18, bi za zapis datoteke uporabili format $dan.mesec.leto\_ure : minute : sekunde.h264$, kjer h264 predstavlja format videa. Knjižnica nam omogoča zapis v takem formatu. Ko ustvarimo ime, uporabimo funkcijo kamere za snemanje.

Končna koda bo izgledala tako:

```
from gpiozero import MotionSensor
from picamera import PiCamera
from datetime import datetime

kamera = PiCamera()
senzor = MotionSensor(2)
```

```
while True:
        senzor.wait_for_motion()
        imeposnetka = datetime.now().strftime
                ("%d.%m.%Y_%H:%M:%S.h264")
        kamera.start_recording(imeposnetka)
        senzor.wait_for_no_motion()
        kamera.stop_recording()
```

Shrani in zaženi svoj program. Ko se ti zdi, da je posnel nekaj posnetkov, ga lahko zaključiš.

4. Ustvarjene posnetke si lahko ogledaš s klikom na terminal. Vanj zapišeš:

```
cd Desktop
omxplayer <imeposnetka> -o hdmi
```

Ukaz:

```
omxplayer 15.8.2016:12.12.h264 -o hdmi
```

predvaja posnetek z imenom datoteke 15.8.2016 : 12.12.h264.

Nabor funkcij za delo s kamero in senzorjem	
PiCamera	
$start.preview()$	vključi predogled
$stop\_preview()$	izključi predogled
$capture()$	funkcija, ki zajame sliko
$rotation$	omogoča rotacijo kamere
$start\_recording()$	kot argument sprejme pot in ime datoteke, kamor shranjuje posnetek
$stop\_recording()$	zaključi snemanje
MotionSensor	
$motion\_detected()$	funkcija, ki zazna premikanje; uporablja se jo v kombinaciji s stavkom if
$wait\_for\_motion()$	čaka na premikanje
$wait\_for\_no\_motion()$	čaka, da se premikanje zaključi

Poglavje 5

Zaključek

V diplomski nalogi smo si najprej pogledali računalnik Raspberry Pi, razvoj modelov skozi leta ter njegovo strojno opremo. Kasneje smo si pogledali programski jezik Python, ki je primeren jezik za učenje programiranja starejših otrok, saj se po videzu in uporabi ne razlikuje preveč od ostalih resnih programskih jezikov. Kljub temu pa otroku omogoča pisanje enostavne in lažje berljive kode.

V osrednjem delu smo predstavili naloge za učenje otrok programiranja. Naloge, ki so razdeljene v tri sklope, se med sabo logično povezujejo. V prvem sklopu otroci osvojijo potrebno znanje programskih konstruktov ter funkcij knjižnice *gpiozero*, da lahko s pomočjo mikrokontrolerja Raspberry Pi ustvarijo igro ter se v njej preizkusijo. V drugem sklopu se spoznajo z igro Minecraft; 3D svetom po katerem se premikajo z uporabo osnovnih funkcij knjižnice *mcpi*. V tretjem sklopu se spoznajo z uporabo kamere Raspberry Pi (Picamera) in pasivnega infrardečega senzorja (senzorja PIR). Naloge tretjega sklopa zahtevajo znanje pridobljeno iz prvih dveh sklopov.

Naloge so primerne za otroke med dvanajstim in petnajstim letom starosti, uporabijo pa jih lahko prav tako vsi začetniki, starši in učitelji.

Zbirko nalog bi lahko uporabili na Poletni šoli, krožku programiranja oz. katerikoli drugi delavnici za otroke, namenjeni učenju programiranja.

V nadaljevanju bi del nalog s svetom Minecrafta lahko razširili ter v njem

sprogramirali igro.

Literatura

[1] A tiny computer attracts a million tinkerers. \http://www.nytimes.com/2013/01/31/technology/personaltech/raspberry-pi-a-computer-tinkerers-dream.html?_r=1. [Dostopano: 28. 8. 2016].

[2] Raspberry Pi becomes best selling British computer. \https://www.theguardian.com/technology/2015/feb/18/raspberry-pi-becomes-best-selling-british-computer. [Dostopano: 28. 8. 2016].

[3] Raspberry Pi Foundation. https://www.raspberrypi.org/. [Dostopano: 16. 8. 2016].

[4] Jason R Briggs. *Python for kids: A playful introduction to programming.* no starch press, 2013.

[5] Thorin Klosowski. How the Raspberry Pi 3 Benchmarks Against Older Models. \http://lifehacker.com/how-the-raspberry-pi-3-benchmarks-against-older-models-1762417275. [Dostopano: 28. 8. 2016].

[6] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman in Evelyn Eastmond. The scratch programming language and environment. *Trans. Comput. Educ.*, 10(4):16:1–16:15, November 2010.

[7] Ben Nuttall. GPIO Zero: Developing a new friendly Python API for Physical Computing. http://bennuttall.com/

gpio-zero-developing-a-new-friendly-python-api-for-physical-computing/. [Dostopano: 2. 9. 2016].

[8] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA, 1980.

[9] Carrie Anne Philbin. *Adventures in Raspberry Pi*. John Wiley & Sons, 2015.

[10] Eben Upton in Gareth Halfacree. *Meet the Raspberry Pi*. John Wiley & Sons, 2012.

[11] Eben Upton in Gareth Halfacree. *Raspberry Pi user guide*. John Wiley & Sons, 2014.