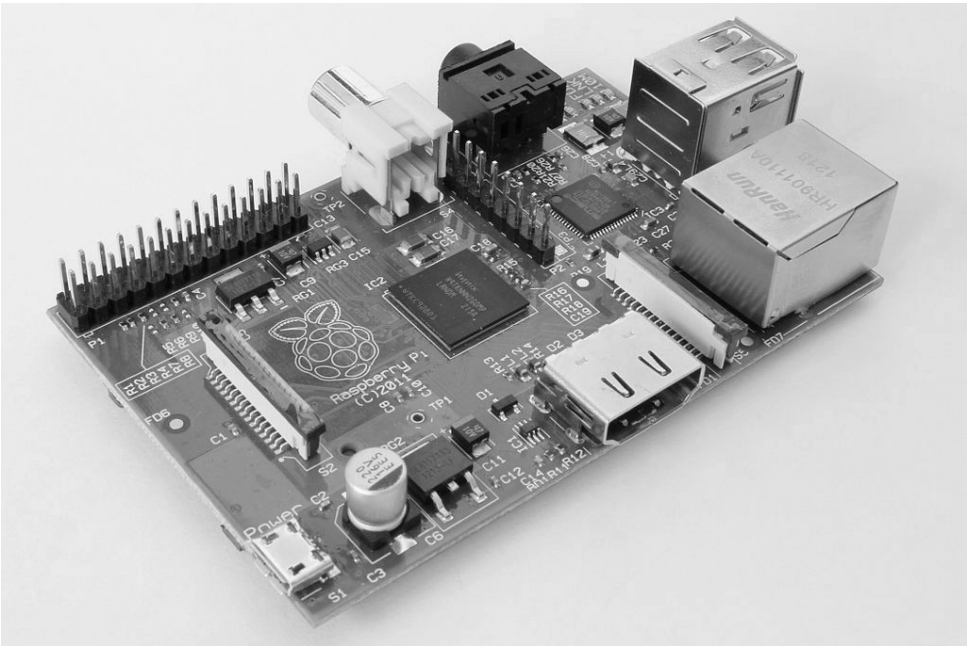
CE

TARTALOMJEGYZÉK

1 Az operációs rendszer telepítésétől kezdve az első Python-programig.

Szinte nincs olyan elektronikai készülék ebben az árkategóriában, amelyről annyit beszéltek volna, mint a Raspberry Pi-ról. A Raspberry Pi, bár első pillantásra egyáltalán nem látszik rajta, teljes értékű számítógép mintegy hitelkártya méretben – nagyon kedvező áron. Nemcsak a hardver olcsó, hanem a szoftver is: az operációs rendszer és az összes mindennapi alkalmazás ingyenesen letölthető az interneten.



1.1 ábra 1.1: A Raspberry Pi – egy miniformátumú számítógép

A speciálisan ráigazított, grafikus felületű Linux-al a Raspberry Pi egy áramtakarékos, hangtalan számítógép-pótlék. A szabadon programozható GPIO-interfésze különösen a hardverbarkácsolók figyelmét kelti fel a Raspberry Pi iránt.

1.1 Mire van szüksége?

A Raspberry Pi aprócska mivolta ellenére teljes értékű számítógép. Ahhoz, hogy használhassa is, szüksége van még- mint a »normális« számítógépeknél is, néhány tartozékra – egy operációs rendszerre, tápegységre, számítógépes hálózatra, monitorra, tasztatúrára és különféle csatlakozókábelekre.

1.1.1 Mikro-USB-mobiltelefon-töltő

A Raspberry Pi számára megfelel minden modern mobiltelefon-töltő. Az USB-töltéstechnika kezdeti idejéből származó régebbi töltőkészülékek azonban túl gyengék. Ha teljesítményéhes USB-készülékeket, például saját tápegységrész nélküli hordozható merevlemezt csatlakoztat, erősebb tápegységre van szüksége. A hálózati tápegységnek 5 V-ot és legalább 700 mA-t, de még inkább 1.000 mA-t kell szolgáltatnia. A beépített teljesítményszabályzó megakadályozza az »áthevülést« túl erős tápegység esetében.

Miben nyilvánul meg, hogy gyenge a tápegység?

Ha ugyan a Raspberry Pi indítja is az operációs rendszert, de azután nem mozgatható a kurzor, vagy a rendszer nem reagál a tasztatúra beadásaira, az gyenge tápáramellátásra utal. Ha nem fér hozzá a csatlakoztatott USB-ceruzához vagy a külső merevlemezes egységhez, alkalmazzon erősebb hálózati tápegységet.

1.1.2 Memóriakártya

A memóriakártya tölti be a Raspberry Pi-ban a merevlemez szerepét. Az tartalmazza az operációs rendszert. Saját adatai és a telepített programok is rajta vannak tárolva. A memóriakártya kapacitásának legalább 4 GB-nek kell lennie, és a gyártó adatai szerint legalább a 4. osztályú szabványt kell támogatnia. Ez a szabvány adja meg a memóriakártya sebességét. Egy korszerű Class-10-memóriakártya nagyobb teljesítménye jól észrevehető.

1.1.3 Tasztatúra

Bármilyen járatos, USB-csatlakozóval ellátott tasztatúra használható. A vezetéknélküli tasztatúrák néha nem működnek, mert túl nagy áramra vagy speciális meghajtóra van szükségük. Ha nem áll rendelkezésére más tasztatúra, akkor egy külön tápáramellátással bíró USB-hubra van szüksége a rádióátvitelű tasztatúra táplálásához.

1.1.4 Egér

USB-csatlakozóval rendelkező egérre csak akkor van szüksége, ha a Raspberry Pi készüléken egy grafikus felhasználói felületű operációs rendszert futtat. Némelyik tasztatúrán vannak kiegészítő USB-hüvelyek egerek számára, úgyhogy nem kell lefoglalnia más csatlakozót a készüléken. Ezek a hüvelyek pl. USB-ceruzához is alkalmazhatók.

1.1.5 (Számítógépes) hálózati kábel

A helyi hálózat routerével való összeköttetéshez egy hálózati kábelre van szükség. Az első felszereléshez mindenesetre még szükség van rá, később használhatja a WLAN-t. Internet-hozzáférés nélkül a Raspberry Pi sok funkcióját nem lehet ésszerűen kihasználni.

1.1.6 HDMI-kábel

A Raspberry Pi HDMI-kábelen keresztül csatlakoztatható monitorra vagy TV-készülékre. DVI-csatlakozós számítógépmonitorra való csatlakozás céljára speciális HDMI-kábeladapter szerezhető be.

1.1.7 Audiokábel

3,5 mm-es jack-dugóval szerelt audiokábel segítségével fejhallgatót vagy számítógép-hangszórót csatlakoztathat a Raspberry Pi-ra. Az audiojel a HDMI-kábelen is rendelkezésre áll. HDMI-hüvellyel bíró TV-készülékhez vagy monitorhoz nincs szükség audiokábelre. Ha egy számítógép-monitort HDMI-kábellel köt össze a DVI-adapterrel, az audiojel ezen a helyen többnyire elvész, úgyhogy ismét szüksége van az analóg audiokimenetre.

1.1.8 Sárga FBAS-videokábel

Ha nem áll rendelkezésre HDMI-monitor, a Raspberry Pi csatlakoztatható egy klasszikus TV-készülékre egy tipikus sárgadugós analóg FBAS-videokábellel is, ekkor azonban a képfelbontás nagyon kicsi lesz. A sárga FBAS-bemenet nélküli TV-készülékek esetében "FBAS-ről SCART-ra" adaptert kell beszerezni. A grafikus felületetet analóg TV-feloldással csak erős korlátozásokkal lehet kezelni.

1.2 A Raspbian-operációs rendszer telepítése

A Raspberry Pi operációs rendszer nélkül kerül szállításra. A hagyományos számítógépektől eltérően, amelyeknek a többsége Windowst alkalmaz, a Raspberry Pi számára egyedileg hozzáigazított Linux operációs rendszert ajánlunk. A Windows egyáltalán nem is futna ezen a takarékos hardveren.

A gyártó által a Raspberry Pi számára ajánlott és támogatott Linux-disztribúció neve Raspbian. A Raspbian az egyik legismertebb Linux-disztribúción, a Debian-Linuxon alapul, amely többek között a népszerű Ubuntu és Knoppix Linux-változat alapja is. Ami a számítógépnél a merevlemez, az a Raspberry Pi esetében egy memóriakártya. Ezen van tárolva az operációs rendszer és az adatok, erről a memóriakártyáról tölti be a Raspberry Pi az operációs rendszert.

1.2.1 A memóriakártya előkészítése a számítógépen.

Mivel a Raspberry Pi még nem tudja betölteni az operációs rendszert, előkészítjük a memóriakártyát a számítógépen. Ehhez szükség van egy kártyaolvasóra a számítógépen. Ez lehet fix bépítésű vagy USB csatlakozású.

A legjobb újonnan gyártott memóriakártya alkalmazása, mivel azokat a gyártó már optimálisan formatálta. De alkalmazhat olyan memóriakártyát is, amelyet előzőleg már használt egy digitális kamerában vagy valamilyen más készülékben. Az ilyen memóriakártyát a Raspberry Pi-hoz való használat előtt újra formatálni kell. Elméletileg ehhez használhatja a Windows formatálási funkcióját. Sokkal jobb azonban az SD Association »SDFormatter« nevű szoftvere . Általa optimális teljesítményűre formatálhatja a memóriakártyát. Ezt az eszközt (tool) díjmentesen letöltheti a *www.sdcard.org/downloads/formatter_4* web-oldalról.

Ha a memóriakártyán vannak partíciók egy korábbi, a Raspberry Pi számára telepített operációs rendszer számára, az SDFormatter-ben nem kerül megjelenítésre a teljes méret. Ebben az esetben alkalmazza a *FULL (Erase) [TELJES (törlés)]* formatálási opciót, és kapcsolja be a *Format Size Adjustment (formátumméret állítása)* opciót. Ezáltal a memóriakártya particionálása újra megtörténik.

A memóriakártya törlődik.

A legjobb, ha egy üres memóriakártyát használ az operációs rendszer telepítésére. Ha adatok lennének a memóriakártyán, ezek az operációs rendszer telepítése közbeni újraformatálás következtében visszavonhatatlanul törlődnek.

1.2.2 A szoftver-telepítő NOOBS

A »New Out Of Box Sofware« (NOOBS) egy különösen egyszerű telepítő a Raspberry-Pi operációs rendszere számára. Itt a korábbiakkal ellentétben nem kell a felhasználónak képi eszközökkel és behúzó blokkokkal bajmolódnia ahhoz, hogy egy betöltésre alkalmas memóriakártyát összehozzon. A NOOBS különféle operációs rendszerek közötti választást ajánl, amikoris az első indításkor közvetlenül a Raspberry Pi-on kiválaszthatjuk a kívánt operációs rendszert, amelyet aztán betöltésre alkalmas formában telepíthetünk a memóriakártyára. Töltse le a kb. 1,2 GB méretű telepítőarchívumot a NOOBS számára a *www.raspberrypi.org/downloads* hivatalos letöltőoldalról, és a számítógépen tömörítse ki egy legalább 4 GB kapacitású memóriakártyára.

Indítsa el most a Raspberry Pi-t ezzel a memóriakártyával. Dugja fel ehhez a dugaszkártyára a Raspberry Pi-t, és csatlakoztassa hozzá a tasztatúrát, az egeret, a monitort és a hálózati kábelt. Utoljára jön az USB-tápcsatlakozás. Ezáltal bekapcsolta a Raspberry Pi-t. Nincs külön bekapcsológomb.

Néhány másodpercre megjelenik egy menü, amelyből kiválaszthatja a kívánt operációs rendszert. Mi a Raspberry-Pi-alapítvány által ajánlott Raspbian operációs rendszert alkalmazzuk.

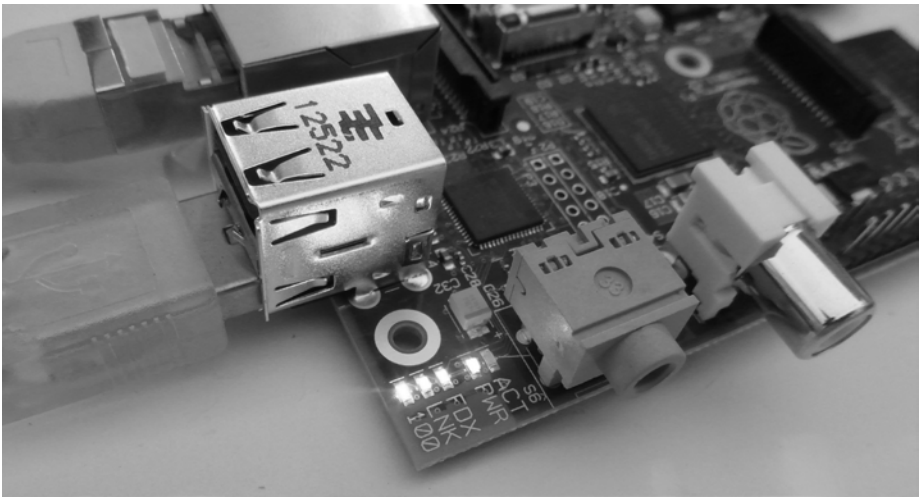
Válassza ki egészen alul a telepítés nyelvét (pl. német), és jelölje meg az előzőleg kiválasztott Raspbian operációs rendszert. A biztonsági kérdésre adott megerősítő válasz után, hogy a memóriakártya felülírásra kerülhet, megkezdődik a telepítés, amely néhány percig tart. Telepítés közben rövid információk jelennek meg a Raspbianról.

1.2.3 A Raspberry Pi LED-jei.

A Raspberry Pi egyik sarkában öt státuszjelző LED helyezkedik el. Az újabb és a régebbi Raspberry-Pi típusok jelölései részben eltérőek, a funkciók azonban azonosak.

Új áramköri kártya (Rev.	Nyomtatott áramköri lap	szín	a LED jelentése
ACT	OK	zöld	hozzáférés a memóriakártyához
PWR	PWR	piros	a tápáramforrással összekötve
FDX	FDX	zöld	a LAN teljes duplex üzemmódban
LNK	LNK	zöld	hozzáférés a LAN-hoz
100	10M	sárga	10/100 Mbit/s sebességű LAN

1.1 táblázat 1.1: A Raspberry Pi LED-jei



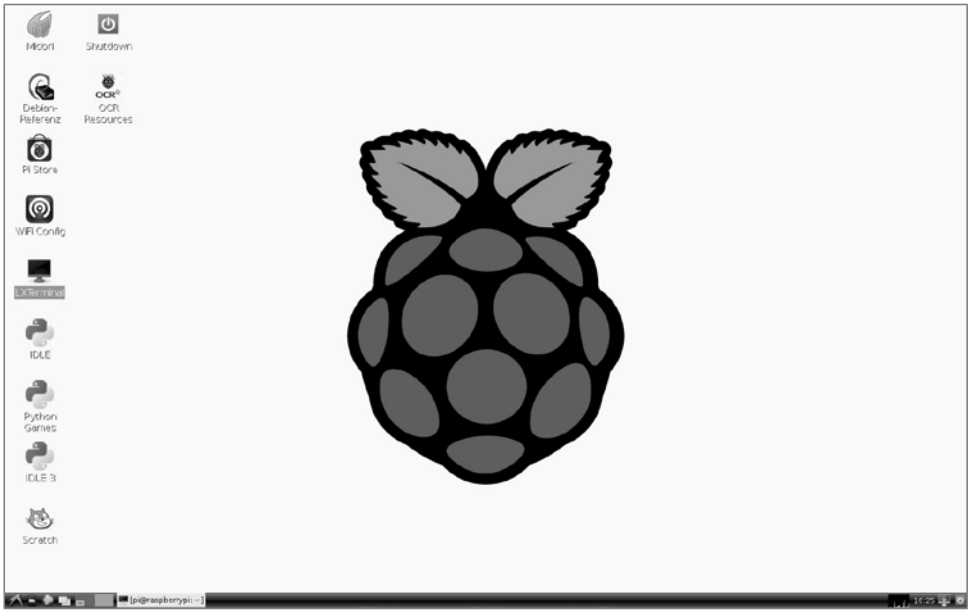
1.2 ábra 1.2: A Raspberry Pi státusz-LED-jei

1.2.4 A Raspberry Pi első indítása

A befejezett telepítés után a Raspberry Pi újra betölti az operációs rendszert, és automatikusan elindítja a raspi-configkonfigurációs eszközt. Itt csak választania kell az *Enable Boot to Desktop (engedélyezve az operációs rendszer betöltése az asztalra)* és a *Desktop Log in as user 'pi' bejelentkezés az asztalon 'pi' felhasználóként)* opció között. A kiválasztott (német) nyelv és tasztatúra-kiosztás (német) a többi fontos alapbeállításhoz hasonlóan már automatikusan kiválasztódott. Újraindítás után a grafikus LXDE-asztal áll rendelkezésre.

1.3 Majdnem olyan, mint a Windows – az LXDE grafikus felület

Sokan már a »Linux« szó láttán megrettennek, mivel attól félnek, hogy kódolt parancssorozatokat kell beírniuk parancssorokba, mint 30 évvel ezelőtt a DOS-ba. Nagy tévedés! A Linux nyitott operációs rendszerként lehetőséget nyit a szoftverfejlesztők számára, hogy szabadon fejlesszenek saját grafikus felületeket. Így a még lényegében parancssor-orientált operációs rendszer felhasználójaként nem vagyunk egy felülethez rögzítve.



1.3. ábra 1.3: A Raspberry Pi LXDE-asztala nagyon hasonlít a Windows XP-ére.

A Raspberry Pi-on futó Raspbian-Linux egy LXDE (Lightweight X11 Desktop Environment) felületet alkalmaz, amelynek egyrészt nagyon kevés rendszer-erőforrásra van szüksége, másrészt startmenüjével és fájlkezelőjével nagyon hasonlít a megszokott Windows-felülethez.

Linux-bejelentkezés

Még a Linuxra jellemző felhasználói bejelentkezés is a háttérben történik meg. Ha egyszer mégis szüksége lenne rájuk: a felhasználónév "pi" és a jelszó "raspberry".

Az egészen balra lent látható LXDE-szimbólum nyitja meg a startmenüt, a mellette lévő két szimbólum pedig a fájlkezelő és a web-böngésző. A startmenü a Windows-éhoz hasonlóan többfokozatú. A gyakran használt programokat a jobb egérgombbal való rákattintással az asztalra ki lehet tenni. Már ott van néhány gyárilag telepített program, a Midori web-böngésző, a Python fejlesztőkörnyezetek és a Pi Store.

A Raspberry Pi kikapcsolása

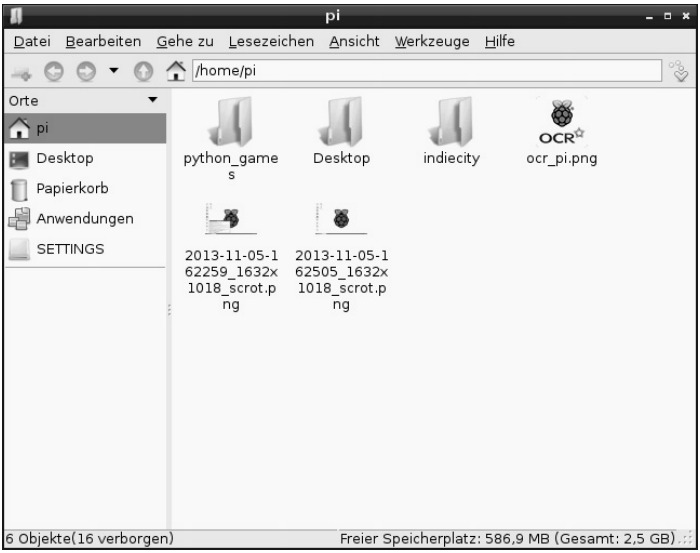
Elméletileg egyszerűen kihúzza a hálózati dugóját, és ezzel kikapcsolódik. Jobb azonban a rendszert egy számítógéphez hasonlóan kikapcsolni. Kattintson ehhez kétszer az asztalon a Shutdown szimbólumra.

1.3.1 Saját fájlok tárolása a Raspberry Pi-on.

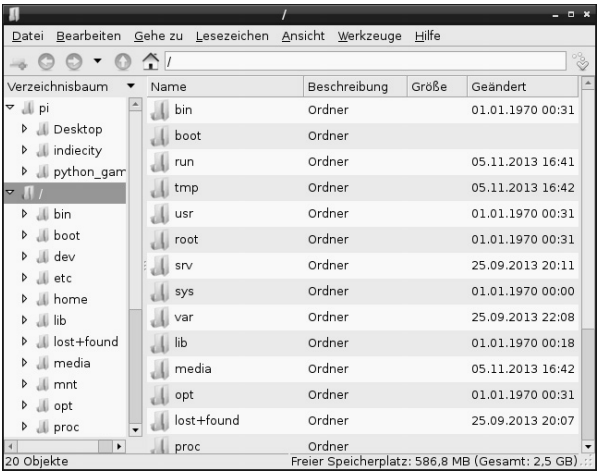
Bár a fájlkezelő a Linux alatt egy kicsit másképp fut, mint a Windows alatt, de nem nehezebb használni. A Raspbian magával hoz egy fájlkezelőt, amely megtévesztésig hasonlít a Windows-Explorerhez. Egy fontos különbség ahhoz képest: a Linux nem választja el szigorúan egymástól a meghajtókat, az összes fájl egy közös fájlrendszerben található.

A Linux alatt az összes saját fájlt alapvetően csakis a sajátkönyvtárba (home directory) helyezi el. A könyvtár neve /home/pi a pi felhasználónév után. A Linux az egyszerű törtvonalat (/) használja a könyvtárszintek elválasztására, és nem a Windows által használt fordított törtvonalat (\ = backslash). Ebben a könyvtárban lesznek tárolva a Python-programjai is. A fájlkezelő, amelyet a Start menün kívül egyébként a Windowshoz hasonlóan a [Win]+[E] nyomógomb-kombinációval is el lehet indítani, normál esetben csak ezt a sajátkönyvtárat (Home) jelzi ki. Néhány program abban automatikusan alkönyvtárakat helyez el.

Ha valaki tényleg mindent akar látni, még azokat a fájlokat is, amelyek különben nem tartoznak a normál felhasználóra, a fájlkezelőt balra fent átkapcsolja az *Orte* (helyek) opcióról a *Verzeichnisbaum* (könyvtárfa) opcióra. Majd az *Ansicht* (nézet) alatti menüben kiválasztja a *Detailansicht* (részletes nézet) opciót, és a képernyő úgy fog kinézni, ahogy az a Linuxtól elvárható.



1.4 ábra 1.4: Így nézhet ki a fájlkezelő a Raspberry Pi-on ...



1.5 ábra 1.5: ... vagy így.

Mennyi szabad hely van még a memóriakártyán?
Nemcsak a számítógépek merevlemeze van állandóan megtelve - a Raspberry Pi memóriakártyája még sokkal hamarabb megtelhet. Annál fontosabb tehát, hogy állandóan a szemünk előtt legyen az, hogy mennyi szabad hely van még a memóriakártyán, és mekkora hely foglalt már. A fájlkezelőnek az ablak alsó szélén lévő státuszsora mutatja jobbra a szabad

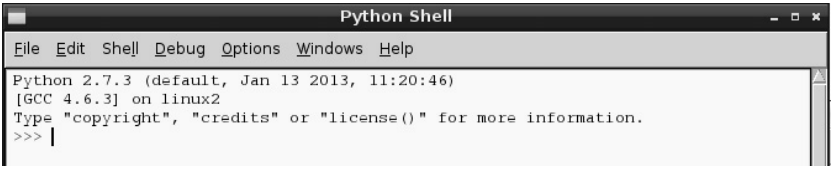
1.4 Az első program a Pythonnal

A Raspberry Pi-on való programozás elkezdéséhez gyárilag telepítve van rá a Python programozási nyelv. A Python meggyőző a tiszta struktúrája miatt, amely a programozás egyszerű elsajátítását teszi lehetővé, de ugyanakkor ideális nyelv is ahhoz, hogy valamit "amúgy gyorsan" automatizáljunk, mert különben kézzel kellene elvégeznünk. Mivel nem kell gondolni semmilyen változódeklarációra, típusra, osztályra, vagy komplikált szabályokra, kész élvezet lesz a programozás.

Python 2.7.3 vagy 3.3.0?

A Raspberry Pi-on gyárilag egyszerre két Python-verzió van telepítve. A legújabb 3.x Python-verzió sajnos könyvtárként más szintaxist alkalmaz, mint a bevált 2.x verzió, emiatt az egyik verzióban írt programok nem futnak a másikkal. Néhány fontos könyvtár, például a játékok és általában a grafikus megjelenítések programozására szolgáló jól ismert PyGame, nem kapható még a Python 3.x-hez. Emiatt, és mert a legtöbb, az interneten hozzáférhető program a Python 2.x számára íródott, ebben a könyvben a jól bevált Python 2.7.3. verziót alkalmazzuk. Ha Raspberry Pi készülékén egy régebbi, 2.x verziószámú Python-verzió van telepítve, példáink ugyancsak működnek vele.

A Python 2.7.3 verziót az asztalon lévő *IDLE* szimbólummal lehet elindítani. Itt az első pillantásra egyszerű beadási ablak jelenik meg egy parancsprompttal.



1.6. ábra 1.6: A Python-Shell beadási ablaka.

Ebben az ablakban nyithatja meg a meglévő Python-programokat, írhat újakat, vagy interaktív módon kidolgozhat közvetlen Python-parancsokat anélkül, hogy saját programot kellene írnia. Adja be például
a promptba a következőt:

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

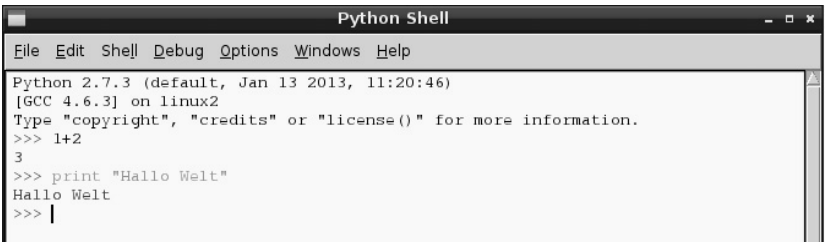
Ekkor azonnal megjelenik a helyes válasz:

```
3
```

Ezen a módon a Python komfortos zsebszámológépként alkalmazható, aminek azonban semmi köze sincs a programozáshoz. A szokott módon a programozási tannfolyamok egy *Hallo Welt*-programmal kezdődnek, amely a képernyőre a »Hallo Welt« szöveget írja ki. Ez a Pythonban olyannyira egyszerű, hogy nem éri meg saját címet adni neki. Adja be csak a Python-Shell-ablakba az alábbi sort:

```
>>> print "Hallo Welt" (nyomtassa ki "halló, világ").
```

Ez az első »program« ezután kiírja a képernyő következő sorába: Hallo Welt .



1.7 ábra 1.7: »Hallo Welt« a Pythonban (felette még látható a számítás eredménye).

Itt egyúttal azt is látjuk, hogy a Python-Shell az érthetővé tétel érdekében automatikusan különböző szövegszíneket alkalmaz. A Python-parancsok narancsszínűek, a karakterláncok zöldek, míg az eredmények kékszínűek. Később még további színeket is fel fog fedezni.

Python-flashcardok (flash háttértárolók)

A Python az ideális programozási nyelv a programozás kezdeteinek az elsajátítására. Csupán a szintaxist és a layout-szabályokat kell kissé megszokni. A mindennapi programozás segítésére a legfontosabb szintaxis-elemeket röviden kis "puskák" formájában ismertetjük. Ezek David Whale Python-Flashcardjain alapulnak. Hogy pontosan miről is van szó, megtudhatja a *bit.ly/pythonflashcards*. web-oldalról. Ezek a kártyák nem a műszaki hátteret magyarázzák meg, hanem csak nagyon rövid példákon keresztül a szintaxist írják le, azaz hogyan készítünk valamit.

1.4.1 Számjáték a Pythonnal

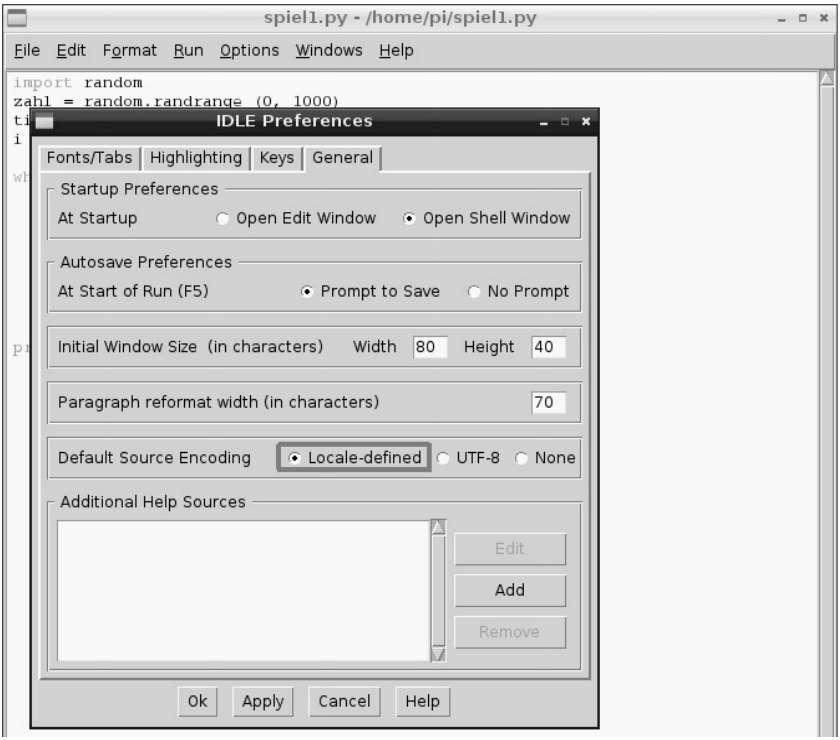
Ahelyett, hogy feltartsuk magunkat a programozás elméletével, algoritmusokkal és adattípusokkal, írjuk meg rögtön az első kis játékunkat a Pythonnal, egy egyszerű találósdival, amelyben a számítógép által véletlenül kiválasztott számot a játékosnak a lehető legkevesebb lépésben ki kell találna.

1. Válassza ki a Python-Shell menüben a *File/New Window* (fájl/új ablak) pontot. Itt megnyílik egy új ablak, amelybe a következő programkódot kell beírnia:

```
import random
zahl (szám) = random.randrange(1000);
        tipp = 0; i = 0, while (amíg) tipp !=
zahl:
    tipp = input("Dein
    Tipp:" ("a Te tipped"))
        ha a szám < tipp:
        print "A keresett szám kisebb, mint a ",tipp
```

2. Mentse el a fájlt a *File/Save As* alatt a spiel1.py néven. Vagy töltse le a kész programfájlt a *www.buch.cd* web-oldalról, és nyissa meg a Python-Shellt a *File/Open* paranccsal. A forrásszövegben lévő színkódolás automatikusan megjelenik, és segít a beírási hibák megtalálásában.

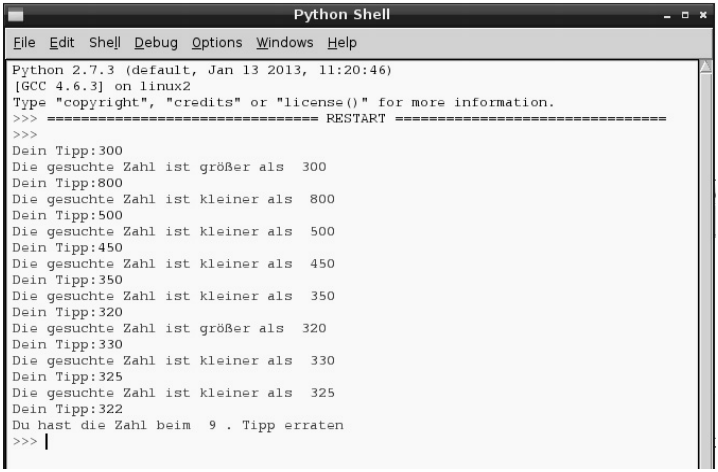
3. Mielőtt még elkezdene játszani, meg kell ismerkednie a német/magyar nyelv egy sajátosságával, ez pedig az "Umlaut"/ékezet. A Python a legkülönfélébb számítógép-platformokat futtatja, amelyek különféle módokon kódolják az umlautokat/ékezeteket. Hogy helyesen jelenjenek meg, válassza ki az *Options/Configure IDLE* menüben, és kapcsolja be a *General (általános)* fülön a *Default Source Encoding (alapértelmezett forráskód)* mezőben a *Locale-defined* (helyileg meghatározott) opciót.



1.8 ábra 1.8: A helyes beállítás az umlautok/ékezetek megjelenítésére a Pythonban.

4. Indítsa el most a játékot az [F5] nyomógombbal vagy a *Run/Run Module*menüponttal.

5. A játék az egyszerűség kedvéért lemond mindenféle grafikus felületről, továbbá a magyarázó szövegekről vagy a beadások plauzibilitására való rákérdezésről. A számítógép a háttérben 0 és 1000 közé eső véletlen számot generál. Adjon be egy tippet, és megtudja, hogy a keresett szám nagyobb vagy kisebb nála. További tippekkel tapogatózzon a helyes szám iránt.



1.9 ábra 1.9: Számjáték a Pythonnal

1.4.2 Ez így működik:

A játék működését azzal próbálhatja ki, hogy játszik vele. Adódik most néhány kérdés: Mi történik a háttérben? Mit jelentenek az egyes programsorok?

import random A véletlenszám generálásához a random nevű külső Python-modul kerül importálásra, amely a véletlengenerátorok különféle funkcióit tartalmazza.

szám = random.randrange(1000) A randrange funkció a random modulból a paraméterek által korlátozott számtartományba, itt 0 és 999 közé eső véletlenszámot generál. A random.randrange() funkció paramétere megadja a véletlenszámok lehetséges darabszámát a 0-val kezdődően, azaz mindig azt az első számot, amelyet nem lehet elérni. Ugyanez érvényes a hurkokra és egyéb Python funkciókra.

Ez a véletlenszám tárolódik a zahl változóban. A változók a Pythonban tetszőleges nevű tárolóhelyek, amelyek számokat, karaktersorozatokat, listákat vagy más adatfajtákat tudnak tárolni.
Némelyik más programozási nyelvtől eltérően ezeket nem kell előre deklarálni.

Hogyan jönnek létre a véletlenszámok?
Általában úgy hiszik, hogy egy programban semmi se történik véletlenül. Hogyan lehetséges akkor egy program számára, hogy véletlenszámokat generál? Ha egy nagy prímszámot valamilyen értékkel elosztunk, akkor az x-edik tizedesjegytől olyan számok jelennek meg, amelyekre már alighanem nem vártunk. Ezek ráadásul bármilyen szabályosság nélkül változnak, ha az osztót állandóan növeljük. Ez az eredmény ugyan látszólag véletlen, azonban reprodukálható egy azonos programmal, vagy ugyanannak a programnak a többszöri lehívásával. Ha azonban most kiveszünk egy, az ezen számok közül néhányból összerakott számot, és elosztjuk egy számmal, amely az aktuális pontos idő másodpercéből vagy a számítógép bármelyik memóriahelyének a tartalmából adódik ki, amely nem reprodukálható, az eredményt emiatt véletlenszámnak nevezzük.

tipp = 0 A tipp változó tartalmazza később azt a számot, amelyre a felhasználó tippel. Az értéke kezdéskor 0.

i = 0 Az i változó a programozók között a programhurkok átfutási számlálójaként honosodott meg. Itt arra alkalmazzuk, hogy a felhasználó által a titkos szám kitalálásához beadott tippek számát számlálja. Kezdéskor ennek a változónak is 0 az értéke.

while tipp != zahl: A while (angolul »amíg«) egy programhurkot vezet be, amely ebben az esetben addig ismétlődik, amíg a tipp, azaz a felhasználó által beadott szám nem egyenlő a titkos zahl számmal. A Python a != karakterkombinációt használja a "nem egyenlő" gyanánt. A kettőspont után következik a tuladonképpeni programhurok.

tipp = input("Dein Tipp:") Az input funkció kiírja a Dein Tipp: (a Te tipped) szöveget, és várja ezután a beadást, amely a tipp változóban tárolódik.

A behúzásoknak fontos szerepe van a Pythonban.
A legtöbb programozási nyelvben a programhurkok vagy a döntések be vannak ugrasztva a margótól, hogy áttekinthetőbbé váljon a programkód. A Pythonban ezek a behúzások nemcsak az áttekinthetőséget javítják, hanem feltétlenül szükség van rájuk a program logikája miatt. Ezért nincs itt szükségünk speciális írásjelekre ahhoz, hogy a hurkokat vagy az eldöntéseket befejezzük.

if zahl < tipp: Ha a zahl titkos szám kisebb mint a felhasználó által beírt tipp szám, akkor ...

```
print "Die gesuchte Zahl ist kleiner als" ("A keresett szám kisebb mint a ",tipp
```

... ez a szöveg kerül kiírásra. A végén itt a tippváltozó áll, hogy a beírt szám a szövegben megjelenjen. Ha ez a feltétel nem teljesül, a beugrasztott sor egyszerűen átugrásra kerül.

if zahl > tipp: Ha azahl titkos szám nagyobb mint a felhasználó által beírt tipp szám, akkor ...

```
print "Die gesuchte Zahl ist größer als " ("A keresett szám nagyobb mint a ",tipp
```

... egy másik szöveg kerül kiírásra.

i += 1 Minden egyes esetben – és ezért már nincs beugrasztva – az iszámláló, amely a kísérletek számát számlálja, eggyel megnövekszik. A += operátort tartalmazó sor ugyanazt jelenti, mint az i = i + 1.

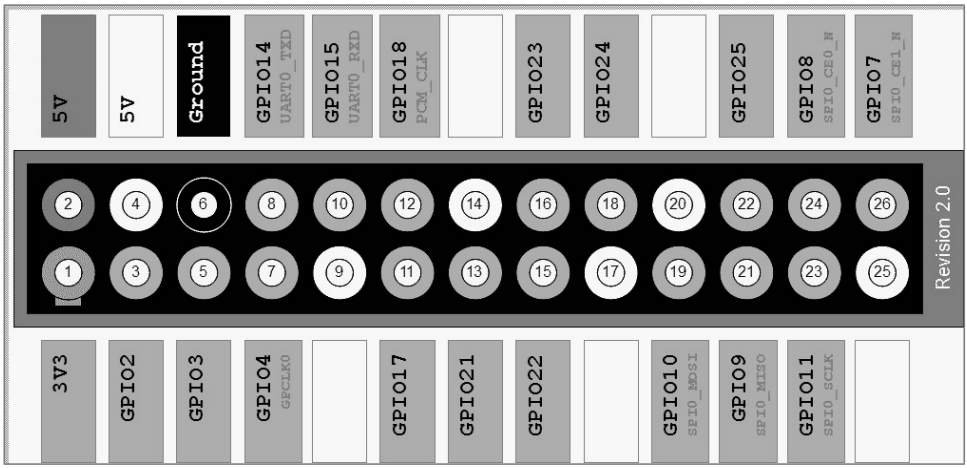
```
print "Du hast die Zahl beim ",i," erraten. ("A számot az ",i," tippnél eltaláltad"
```

Ez a sor még jobban be van ugratva, ami azt jelenti, hogy a while-hurok is befejeződött. Ha a feltétel már nem teljesül, azaz a felhasználó által megtippelt szám tipp már nem más (hanem azonos) a zahltitkos számmal, ez a szöveg kerül kiírásra, amely két mondatrészből és az i változóból van összeállítva, és ezáltal megadja, hogy hány kísérletre volt szüksége a felhasználónak. A Python-programok nem igényelnek saját befejezési utasítást. Csak egyszerűen befejeződnek az utolsó parancs után, ill. egy olyan hurok után, amely már nem kerül ismételten végrehajtásra, és nem követi további parancs.

2 Az első LED világít a Raspberry Pi-on

A Raspberry Pi sarkán elhelyezkedő 26-pólusú érintkezősávon keresztül közvetlenül csatlakoztathatja a hardvereszközöket, pl. nyomógombos bevitelhez vagy LED-ek programvezérelt kigyújtásához. Ennek az érintkezősávnak a megnevezése GPIO. Ez a »General Purpose Input Output« angol kifejezés rövidítése, amelynek a magyar jelentése egyszerűen »Általános bevitel és kiadás«.

Ebből a 26 csapból 17 vagylagosan bemenetként vagy kimenetként programozható, és így sokrétűen használható hardverbővítésre. A többi érintkező a tápáramellátásra és egyéb célokra van fixen bekötve.



2.1 ábra 2.1: A GPIO-interfész bekötése A szürke vonal fent és balra a kártya szélét jelöli. A GPIO 2. csapja teljesen a Raspberry Pi sarkának a szélén helyezkedik el.

[illegible] érintkezőcsap azonban nem köthető össze egy GPIO-bemenettel.

2.1 Alkatrészek a csomagban

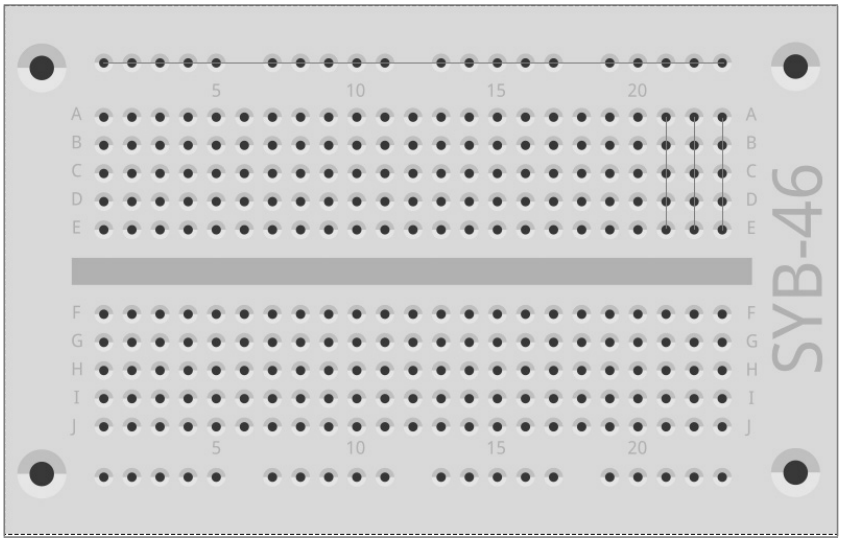
A tanulókészlet különféle elektronikai alkatrészeket tartalmaz, amelyekkel megépíthetők az ismertetett kísérletek (és természetesen saját kísérletei is). Ehelyt csupán röviden ismertetjük az alkatrészeket. A használatukhoz kapcsolódó szükséges gyakorlati tapasztalatokat a

tényleges kísérletek hozzák magukkal.

- 2 db dugasztábla
- 1 db piros LED
- 1 db sárga LED
- 1 db zöld LED
- 1 db kék LED
- 4 db nyomógomb
- 4 db 10 kohm-os ellenállás (barna-fekete-narancs)
- 4 db 1 kohm-os ellenállás (barna-fekete-piros)
- 4 db 220 ohm-os ellenállás (piros-piros-barna)
- 12 db összekötőkábel (dugasztábla – Raspberry Pi)
- kb. 1 m kapcsolóhuzal

2.1.1 dugasztábla

Elektronikai kapcsolások felépítésére két dugasztáblát tartalmaz a csomag. Ezekbe az elektronikai alkatrészek közvetlenül bedughatók a szabvány rasztertávolságban lévő furatokba forrasztás nélkül. Ezeken a dugasztáblákon a szélső hosszanti lyuksorok (X és Y) érintkezői össze vannak kötve egymással.



2.2 ábra 2.2: A csomagban található dugasztábla néhány példaként berajzolt összeköttetéssel.

Ezeket az érintkezősorokat gyakran a kapcsolások tápáramforrásának a pozitív és negatív pólusául használjuk. A többi érintkezősorban öt-öt érintkező (A-tól E-ig és F-től J-ig) keresztben van összekötve egymással, míg a tábla közepén egy hézag van. Így a tábla közepén nagyobb alkatrészeket lehet bedugni, és a széle felé lehet huzalozni.

2.1.2 Összekötőkábelek

A színes összekötőkábelek mindegyikének az egyik oldalán egy kis huzaldugó van, amelyet be lehet dugni a dugasztáblába. A másik oldalukon egy dugaszhüvely van, amely illik a Raspberry Pi-nak a GPIO érintkezőcsapjaira.

Ezenkívül még egy kapcsolóhuzalt is tartalmaz a tanulókészlet. Általa rövid áthidalókat lehet létrehozni a dugasztábla érintkezősorainak az áthidalására. Vágja le kellő hosszúságú darabokra a kapcsolóhuzalt egy oldalcsípőfogó segítségével az egyes kísérletek leírásának megfelelően. Ahhoz, hogy a huzalokat könnyebben be lehessen dugni a dugasztáblába, ajánlatos kissé ferdén levágni őket, hogy egy kis ékalakú végződés jöjjön létre. A huzaldarabok két végéről távolítsa el a szigetelést mintegy fél centiméter hosszban.

2.1.3 Ellenállások és színkódjaik

Az ellenállásokat a digitális elektronikában főleg egy mikrovezérlő portjainak az áramkorlátozására, továbbá a LED-ek előtétellenállásaiként allkalmazzák. Az ellenállás mértékegysége az ohm.
1.000 ohm egy kiloohm, rövidítve egy kohm.

Az ellenállásértéket az ellenállásokon színes gyűrűk jelölik. A legtöbb ellenállásnak négy ilyen színes gyűrűje van. Az első két színes gyűrű a számértéket adja meg, a harmadik egy szorzó, és a negyedik a tűrés. Ez a tűrésgyűrű többnyire arany- vagy ezüstszínű – ezek a színek nem fordulnak elő az első gyűrűkön, úgyhogy egyértelmű a leolvasás iránya. A tűrésértéknek alig van szerepe a digitális elektronikában.

szín	ellenállásérték ohm-ban			
	1. gyűrű (tizes)	2. gyűrű	3. gyűrű (szorzótényez	4. gyűrű (tűrés)
ezüst			$10^{-2} = 0{,}01$	±10%
arany			$10^{-1} = 0{,}1$	±5%
fekete		0	$10^{0} = 1$	
barna	1	1	$10^{1} = 10$	±1 %
piros	2	2	$10^{2} = 100$	±2 %
narancs	3	3	$10^{3} = 1.000$	
sárga	4	4	$10^{4} = 10.000$	
zöld	5	5	$10^{5} = 100.000$	±0,5 %
kék	6	6	$10^{6} = 1.000.000$	±0,25 %
ibolya	7	7	$10^{7} = 10.000.000$	± 0,1 %.
szürke	8	8	$10^{8} = 100.000.000$	± 0,05%
fehér	9	9	$10^{9} = 1.000.000.000$	

2. 1. táblázat 2.1: A táblázat az ellenállásokon lévő színes gyűrűk jelentését mutatja.

A tanulókészletben három különböző értékű ellenállások vannnak:

érték	1. gyűrű (tizes)	2. gyűrű	3. gyűrű (szorzó)	4. gyűrű (tűrés)	Alkalmazás
220 ohm	piros	piros	barna	arany	előtétellenállás a LED-
1 kohm	barna	fekete	piros	arany	védőellenállás a GPIO-
10 kohm	barna	fekete	narancs	arany	lehúzó-ellenállás a GPIO-bemenetekhez

1. táblázat 2.2: A tanulókészletben lévő ellenállások színkódja.

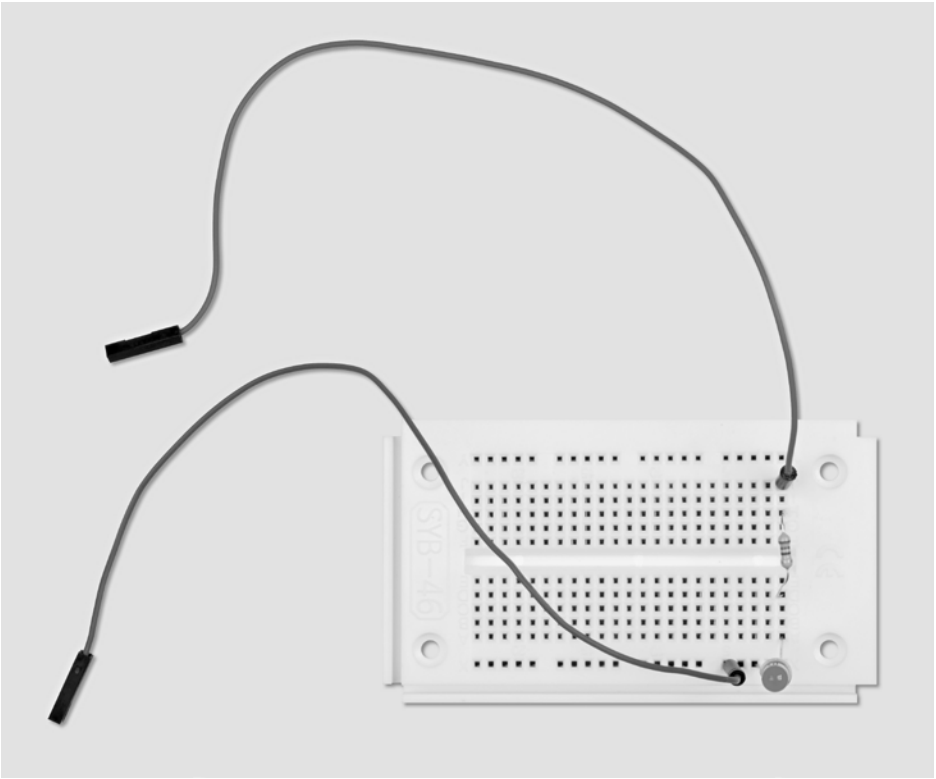
Különösen az 1 kohmos és a 10 kohmos ellenállások színjelöléseire figyeljen pontosan. Ezek könnyen felcserélhetők egymással.

2.2 A LED-ek csatlakoztatása

A GPIO-portokra LED-ek (LED = Light Emitting Diode, magyarul fénydióda/világító dióda) csatlakoztathatók fényjelzések és fényeffektusok keltésére. Ehhez be kell kötni egy-egy 220 ohmos előtétellenállást (piros-piros-barna) az alkalmazott GPIO-érintkező és a LED anódja közé az átfolyó áram korlátozása, és ezáltal a LED leégésének a megakadályozása érdekében. Az előtétellenállás ezenkívül még a Raspberry Pi GPIO-kimenetét is védi, mivel a LED a nyitóirányú áramra majdnem nem ad semmilyen ellenállást, és emiatt a GPIO-port testre kötve hamar túlterhelődhet. A LED katódját a 6. csapon lévő testvezetékkel kell összekötni.

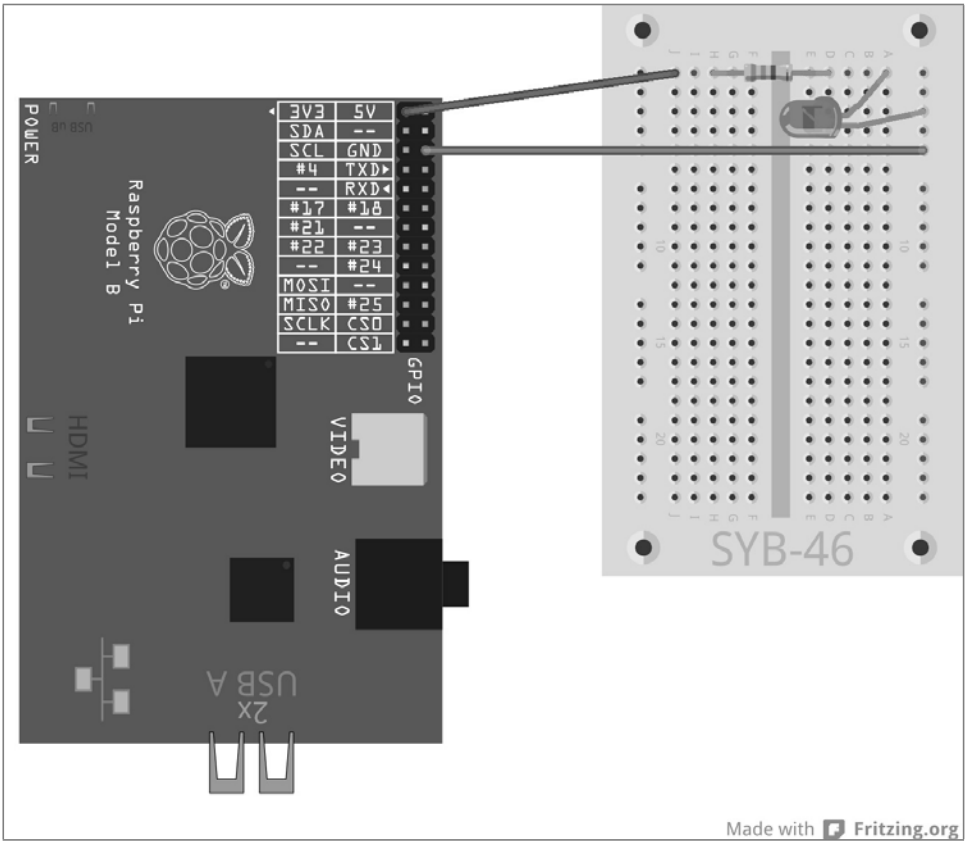
Milyen irányba kell csatlakoztani a LED-et
A LED két kivezetése különböző hosszúságú. A hosszabbik a pozitív pólus, az anód, a rövidebbik a katód. Egyszerű megjegyezni, a plusz jel eggyel több vonalat tartalmaz, mint a mínusz jel, és ezért a kivezetése egy kicsit hosszabb.

Elsőként kössön rá az ábrának megfelelően egy LED-et egy 220 ohmos előtétellenálláson (piros-piros-barna)
keresztül a +3,3 V-os csatlakozócsapra (1.), és kösse rá a LED negatív pólusát a testvezetékre (6.csap).



2.3 ábra 2.3: A dugasztábla beültetése egy LED csatlakoztatásakor.

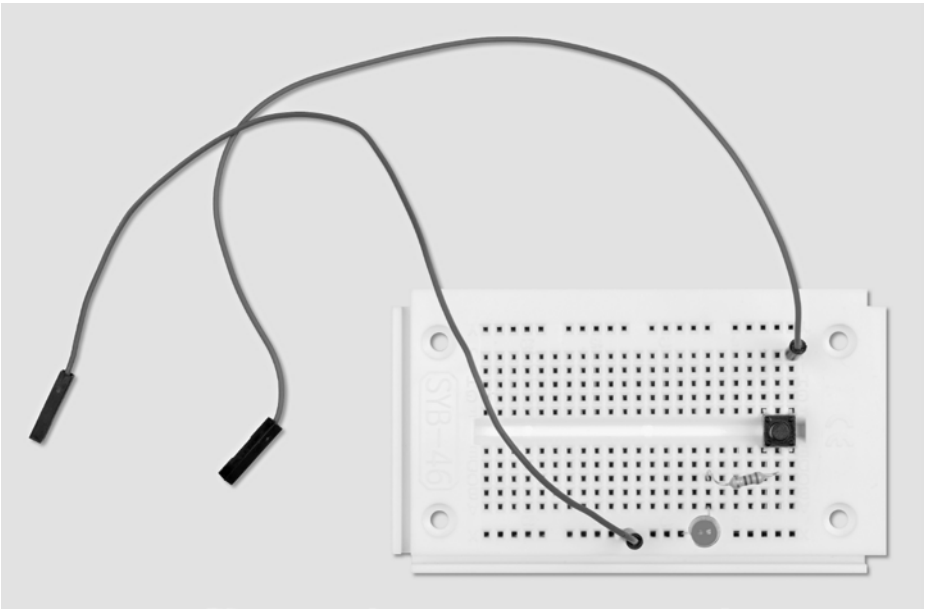
A szükséges
alkatrészek: 1 db
dugasztábla
1 db piros LED
1 db 220-ohmos
ellenállás 2 db
összekötőkábel



2.4 ábra 2.4: Az első LED a Raspberry Pi-on.

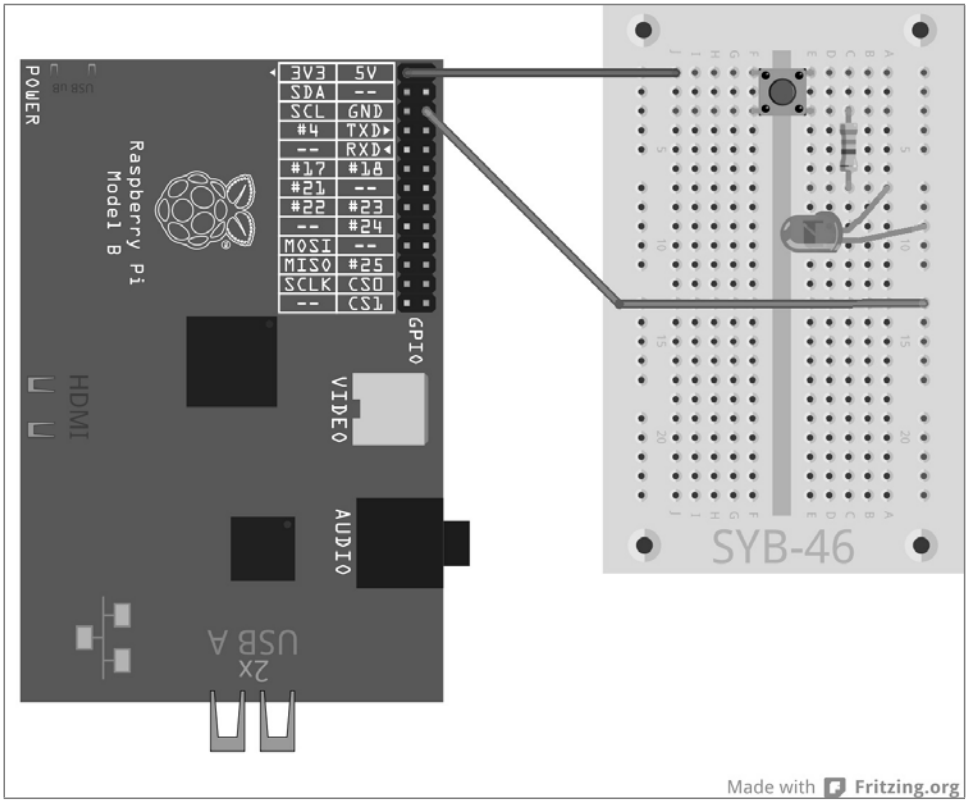
Ebben az első kísérletben a Raspberry csupán a LED tápáramforrásául szolgál. A LED folyamatosan világít, nem kell hozzá szoftver.

A következő kísérletben rakjon be egy nyomógombot a LED bekötővezetékébe. A LED most csak akkor világít, ha megnyomja ezt a nyomógombot. Ehhez se kell szoftver.



2.5 ábra 2.5: A dugasztábla beültetése egy nyomógombbal kapcsolható LED számára.

A szükséges
alkatrészek: 1 db
dugasztábla
1 db piros LED
1 db 220-ohmos
ellenállás 1 db
nyomógomb
2 db összekötőkábel



2.6 ábra. 2.6: A LED nyomógombbal a Raspberry Pi-on.

2.3 A GPIO a Pythonnal

Ahhoz, hogy a GPIO-portokat a Python-programokkal tudjuk használni, telepíteni kell a Python-GPIO-Bibliothek könyvtárat. Ha nem biztos afelől, hogy az összes szükséges modul telepítve van, telepítse az aktuális verziót az alábbi konzolparancsokkal:

```
sudo apt-get update
sudo apt-get install python-dev
sudo apt-get install python-rpi.gpio
```

A GPIO-portok, mint a Linux alatt futó összes készüléknél megszokott dolog, fájlként könyvtárstruktúrába vannak rendezve. Ezekhez a fájlokhoz való hozzáféréshez gyökér-jogosultságra van szükség. Indítsa el tehát a Python-Shellt gyökér-jogosultsággal egy LXT-terminálon keresztül: `sudo idle`

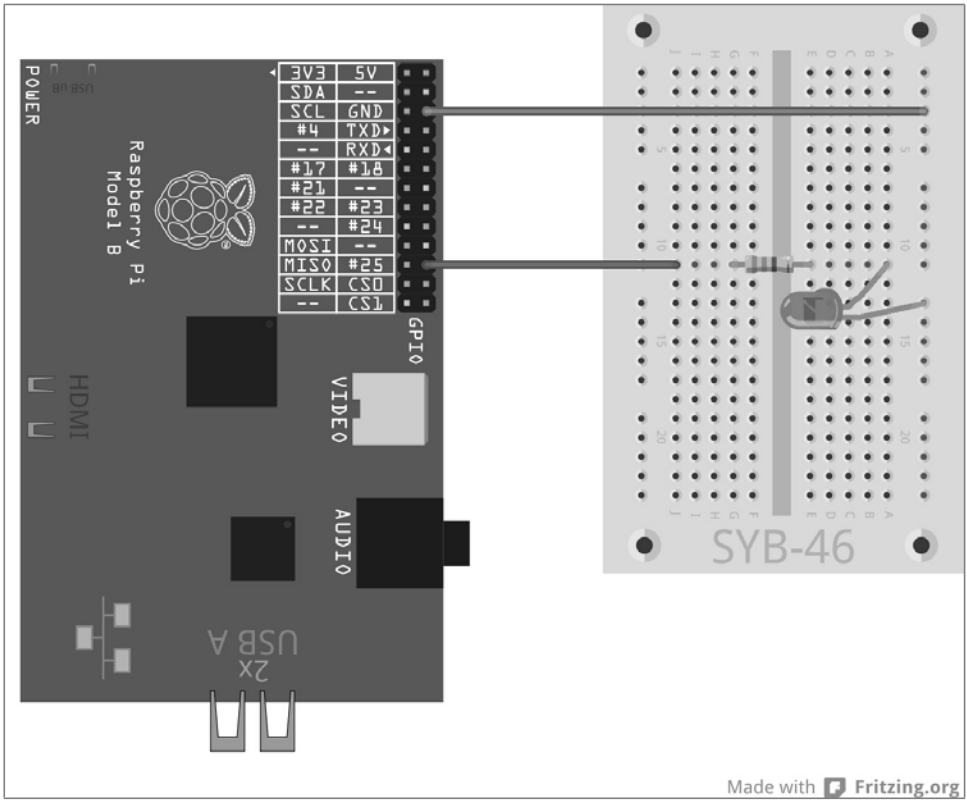
2.4 LED-ek be- és kikapcsolása

Csatlakoztasson a következő ábrának megfelelően egy LED-et egy 220-ohmos előtétellenálláson (piros-piros-barna) keresztül a 25. GPIO-portra (22. csap), és már nem közvetlenül a +3,3 V-os csatlakozópontra, és kösse össze a LED negatív pólusát a dugasztábla testsínjén át a Raspberry Pi testvezetékével (6. csap).

A szükséges alkatrészek: 1 db dugasztábla
1 db piros LED
1 db 220-ohmos ellenállás 2 db összekötőkábel

A következő program, a led.py a LED-et 5 másodpercre bekapcsolja, majd ismét kikapcsolja:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(25, GPIO.OUT)
GPIO.output(25, 1)
time.sleep(5)
GPIO.output(25, 0)
GPIO.cleanup()
```



2.7 ábra 2.7: Egy LED a 25. GPIO-porton

2.4.1 Így működik

A példa az RPi.GPIO-könyvtár alapvető funkcióit mutatja be.

import RPi.GPIO as GPIO Az RPi.GPIO könyvtárat minden olyan Python-programba importálni kell, amelyben alkalmazni akarjuk. Ezzel az írásmóddal a könyvtár összes funkciója a GPIO előtaggal megszólítható.

import time A gyakran alkalmazott time Python-könyvtárnak semmi köze sincs a GPIO-programozáshoz. Az idő- és dátumszámítás funkcióit tartalmazza, többek között a time.sleep funkciót is, amellyel egyszerű módon lehet várakozási időt megvalósítani egy programban.

GPIO.setmode(GPIO.BCM) Minden program elején definiálni kell a GPIO-portok megjelölését. Általában a BCM szabványszámozást alkalmazzuk.

A GPIO-portok számozása

Az RPi.GPIO könyvtár két különböző módszert támogat a portok megjelölésére. A BCM üzemmódban az ismert GPIO-portszámokat alkalmazzuk, amelyeket a parancssorok szintjén vagy a Shell-scriptekben (parancsállományok) is használunk. A másik lehetséges üzemmódban, a BOARD-ban, a megjelölés a Raspberry-Pi-kártya érintkezőcsapjai számozásának felel meg.

GPIO.setup(25, GPIO.OUT) A GPIO.setup funkció egy GPIO-portot kimenetként vagy bemenetként inicializál. Az első paraméter megjelöli a portot az előre meghatározott BCM vagy BOARD üzemmódban a GPIO-számával vagy csapszámával. A második paraméter vagy GPIO.OUT egy kimenet számára, vagy GPIO.IN egy bemenet számára állhat.

GPIO.output(25, 1) Az imént inicializált portra egy logikai 1 kerül kiadásra. Az arra csatlakoztatott LED világít. Az 1 helyett kiadhatók a True (igaz) vagy a GPIO.HIGH előzetesen meghatározott értékek is.

time.sleep(5) A program elején importált time-könyvtárnak ez a funkciója 5 másodperces várakozási időt hoz létre, mielőtt a program továbbfutna.

GPIO.output(25, 0) A LED kikapcsolásakor a 0 , ill. False (hamis), vagy a GPIO.LOW érték jelenik meg a GPIO-porton.

GPIO.cleanup() A program befejezésekor az összes GPIO-portot vissza kell állítani. Ez a sor a program által inicializált összes GPIO-porton egyszerre hajtja végre ezt. A más programok által inicializált portok állapota változatlan marad. Így az egyéb, esetleg párhuzamosan futó programok zavartalanul futhatnak.

GPIO-figyelmeztetések vétele

Ha konfigurálni kell egy olyan GPIO-portot, amely nem lett jól visszaállítva, vagy egy félbeszakadt program még nyitva tartja, figyelmeztetéseket kaphatunk, amelyek azonban nem szakítják meg a program futását. A program fejlesztése közben ezek a figyelmeztetések nagyon hasznosak lehetnek egy hiba felfedezése szempontjából. Egy kész programban azonban egy tapasztalatlan felhasználót összezavarhatnak. Emiatt a GPIO-könyvtár által nyújtott GPIO.setwarnings(False) funkció útján elnyomhatja ezeket a figyelmeztetéseket.

2.5 A Python indítása GPIO-támogatással terminál nélkül

Aki sokat szokott bütykölni a Pythonnal és a GPIO-val, az nem akar minden egyes alkalommal egy LXTerminal-t lehívni az IDLE elindításához. Sokkal egyszerűbb módja is van ennek. Helyezzen el ehhez egy szimbólumot az asztalon, amely a korlátozás nélküli felhasználói felhatalmazással (Superuser) hívja le a Python-IDE-t:

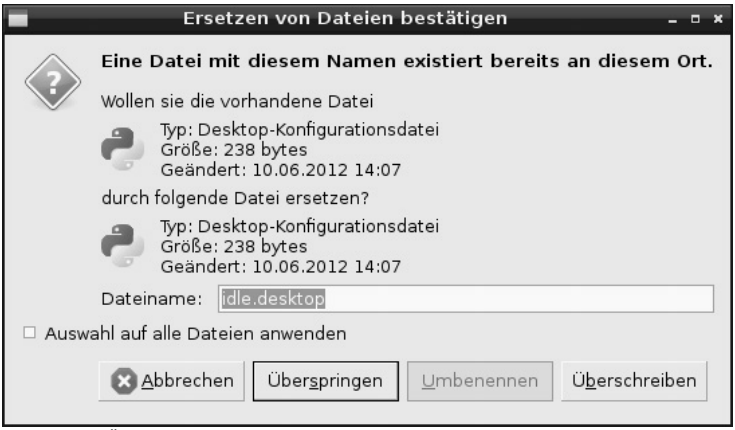
- Helyezze el az előre telepített *IDLE* asztali szimbólum egy másolatát. Ehhez tegye a következőket:

- Kattintson a jobb egérgombbal az asztalon lévő *IDLE* szimbólumra, és válassza ki a felbukkanó menüből a *Kopieren* (másolás) opciót.

2.8 ábra 2.8: *Az IDLE*-asztali szimbólum másolása.

Majd kattintson a jobb egérgombbal az asztalra, és válassza ki a felbukkanó menüben az *Einfügen* (beillesztés) opciót. Mivel van már egy azonos nevű asztali parancsikon, megjelenik egy üzenet a másolat elhelyezésének a megkísérlésekor.

Változtassa meg a másolat *idle.desktop* nevét az *idle_gpio.desktop* névre. Az asztalon lévő szimbólum egyelőre nem változik meg. A kijelzett név az *IDLE* marad.



2.9 ábra 2.9: Üzenet egy asztali parancsikon másolásakor.

Kattintson a jobb egérgombbal az asztalon lévő szimbólum másolatára, és válassza ki a felbukkanó menüben a *Leafpad* opcióra. Az asztali parancsikonok a Linuxban tiszta szövegfájlok, amelyek egy szövegszerkesztővel módosíthatók.

2.10 ábra 2.10: Az asztali parancsikonok a Leafpad szövegszerkesztőben.

Végezze el itt az ábrán látható két változtatást:

- Változtassa meg a Name (név) mezőben lévő szöveget az IDLE GPIO szövegre. Ez a képernyőn megjelenő név.

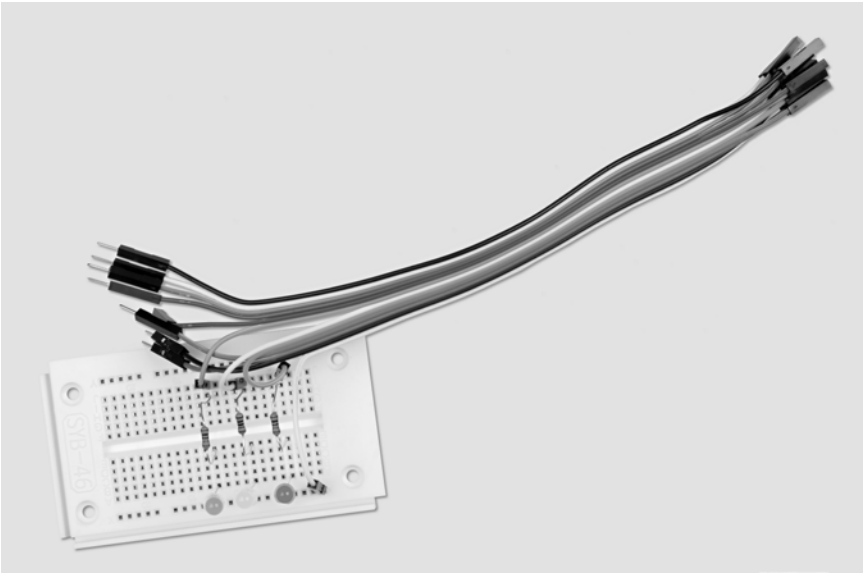
- Rakja be az Exec mezőben a tulajdonképpeni parancslehívás elé a sudo szót.

Zárja be a szövegszerkesztőt, és mentse ki a fájlt. Kattintson kétszer az új asztali parancsikonra, és indítsa el az *IDLE* Python-fejlesztői környezetet (IDE), korlátozás nélküli felhasználói felhatalmazással. Most használni tudja már a GPIO-funkciót anélkül, hogy le kellene hívnia a Pythont egy LXTerminal segítségével.

3 Közlekedési lámpa

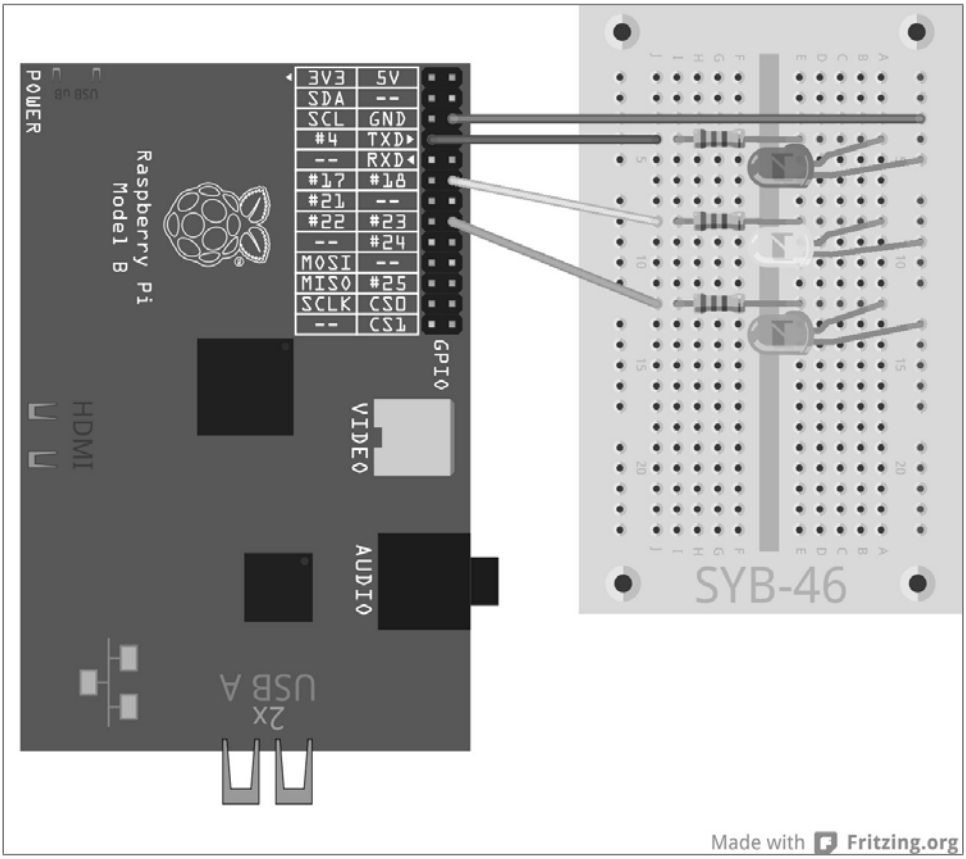
Egyetlen LED bekapcsolása, majd kikapcsolása izgalmas lehet az első pillanatban, de ehhez tulajdonképpen nincs szükség számítógépre. Egy közlekedési lámpa felépítése a tipikus világítási ciklusával a zöld lámpáról a sárgán keresztül a pirosra, majd egy piros-sárga fénykombináción át vissza a zöld lámpára három LED-del könnyen megy, és további programozási technikákat mutat be a Pythonban.

Építse fel az ábrázolt kapcsolást a dugasztáblán. A LED-ek vezérlésére három GPIO-portot és egy közös testvezetéket használunk. A GPIO-portok számozása BCM-üzemmódban az ábrán látható a Raspberry Pi-on.



3.1 ábra 3.1: A dugasztábla beültetése a közlekedési lámpához.

A szükséges alkatrészek: 1 db dugasztábla
1 db piros LED 1 db sárga LED
1 db zöld LED
3 db 220-ohmos ellenállás 4 db összekötőkábel



3.2 ábra 3.2: Egy egyszerű közlekedési lámpa.

Az ampel01.py program vezérli a közlekedési jelzőlámpát:

```
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 ("Strg+C beendet das Programm")
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 Így működik

Az első sorok már ismertek, ezek importálják az RPi.GPIO könyvtárat a GPIO-portok vezérléséhez és a time könyvtárat az időkésleltetéshez. Ezután az előző példához hasonlóan a GPIO-portok számozása a BCM üzemmódra állítódik.

rot = 0; gelb = 1; gruen = 2 Ezek a sorok a három változót rot (piros), gelb (sárga) és gruen (zöld) határozzák meg a három LED-re. Ezáltal nem kell megjegyeznünk a programban számokat vagy GPIO-portokat, hanem a LED-eket egyszerűen a színük alapján lehet vezérelni.

Ampel=[4,18,23] A három LED vezérléséhez egy lista készül, amely a GPIO-számokat abban a sorrendben tartalmazzák, ahogyan a LED-ek elhelyezkednek a dugasztáblán. Mivel a GPIO-portok a programnak csak ezen a helyén szerepelnek, a programot nagyon egyszerűen át lehet alakítani, ha más GPIO- portokat akar használni.

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

A három alkalmazott GPIO-portot egymás után inicializálja kimenetnek. Eközben nem alkalmazunk GPIO-portszámokat, hanem az előzőleg definiált listát. Egy listán belül az egyes elemeket számok jelölik a 0-val kezdődően. Ampel[0] tehát az első elem, jelen esetben 4. A rot, gelb és gruen változó a 0, 1 és 2számot tartalmazzák, amelyekre a lista elemeinek az indexéül van szükség. Ilymódon a GPIO-portokat színekkel lehet megcímezni:

- Ampel[rot] megfelel a 4 GPIO-portnak a piros LED-del.
- Ampel[gelb] megfelel a 18 GPIO-portnak a sárga LED-del.
- Ampel[gruen] megfelel a 23 GPIO-portnak a zöld LED-del.

A GPIO.setup-utasítás tartalmazhat egy initial optionális paramétert, amely a GPIO-porthoz már az inicializáláskor hozzárendel egy logikai állapotot. Általa ebben a programban a zöld LED-et már kezdéskor bekapcsoljuk. A másik két LED a program kezdetén kikapcsolt állapotban van.

print ("Strg+C beendet das program") Most egy rövid kezelési utasítás jelenik meg a képernyőn. A program automatikusan fut. A [Strg]+[C] (angol tasztatúrán [ctrl] + [C]) nyomógomb-kombinációval kell befejezni. Annak a lekérdezésére, hogy a felhasználó befejezi-e a programot a [Strg]+[C] megnyomásával, a try...except-lekérdezést alkalmazzuk. Ekkor a try: alatt bevitt programkód normálisan kivitelre kerül. Ha eközben

rendszerkivétel lépne fel – ami hiba is lehet, vagy akár megszakadt a [Strg]+[C] nyomógomb-kombináció, és az except-utasítás a program végén hajtódik végre.

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

Ezzel a nyomógomb-kombinációval egy KeyboardInterrupt-ot (nyomógombos megszakítás) váltunk ki, és a hurkot automatikusan elhagyjuk. Az utolsó sor lezárja az alkalmazott GPIO-portokat, és ezzel kikapcsolja az összes LED-et. Majd befejeződik a program. A GPIO-portok vezérelt lezárásakor nem jelennek meg se rendszerfigyelmeztetések, se megszakítás-jelentések, amelyek összezavarhatnák a felhasználót. A tulajdonképpeni jelzőlámpaciklus végtelen hurokban fut:

while True : Ezekhez a végtelen hurkokhoz szükség van egy megszakítási feltételre, mivel egyébként soha nem fejeződne be a program.

time.sleep(2) A program kezdetén, és a hurok minden egyes elkezdődésénél a zöld LED 2 másodpercig világít.

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

Most a zöld LED kialszik, és helyette a sárga LED kigyullad. Ez csupán 0,6 másodpercig világít.

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

Most a sárga LED ismét kialszik, és helyette a piros LED gyullad ki. Majd ez csupán 2 másodpercig világít. A közlekedési jelzőlámpa piros fázisa általában lényegesen hosszabb, mint a sárga fázis.

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

A piros-sárga fázis megkezdéséhez még a sárga LED is kigyullad, anélkül, hogy egy másik LED kialudna. Ez a fázis 0,6 másodpercig tart.

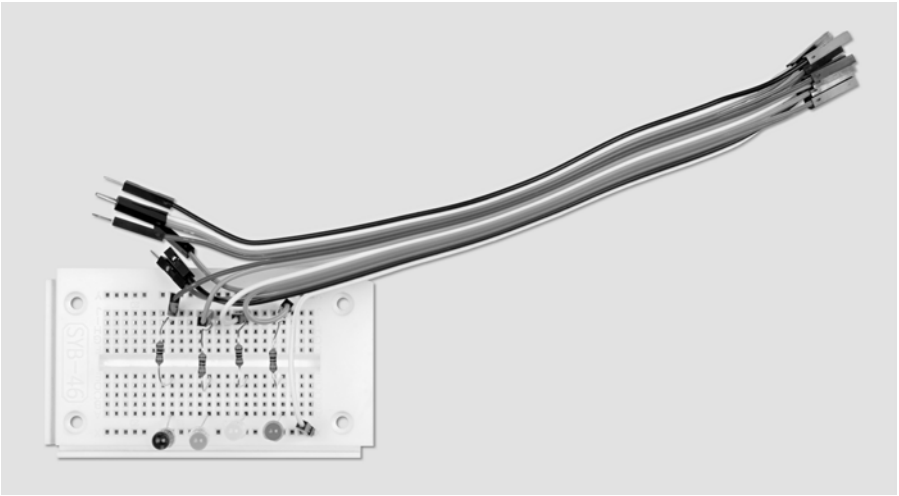
```
        GPIO.output(Ampel[rot],False)
        GPIO.output(Ampel[gelb],False)
        GPIO.output(Ampel[gruen],True)
```

A hurok végén a közlekedési jelzőlámpa ismét zöldre vált. A piros és a sárga LED kialszik, a zöld bekapcsolódik. A hurok a jelzőlámpa zöld fázisában kezdődik újra 2 másodperces várakozási idővel. Természetesen az összes idő tetszés szerint állítható be. A valóságban a közlekedési jelzőlámpa fázisai a kereszteződés méreteitől és a forgalom sűrűségétől függenek. A sárga- és a piros-sárga fázis általában 2-2 másodpercig tart.

4 Gyalogos jelzőlámpa

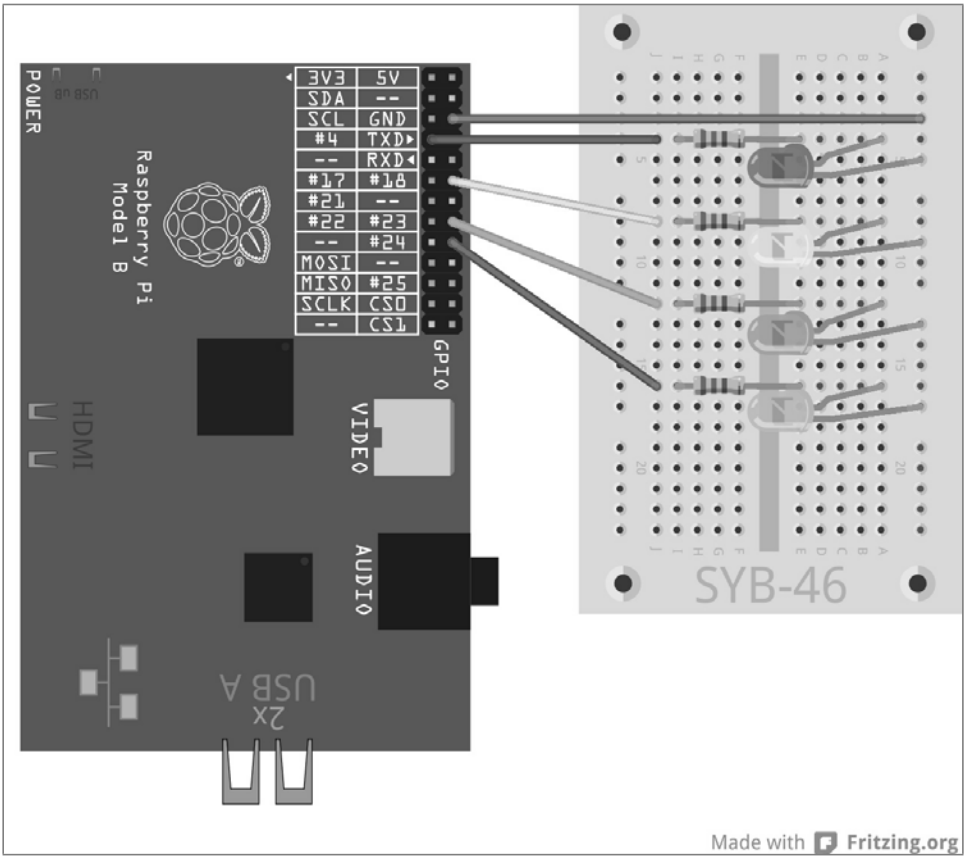
A következő kísérletben kibővítjük a jelzőlámpakapcsolást még egy gyalogos jelzőlámpával, amely a közlekedési lámpa piros fázisa alatt villogó fénnyel jelez a gyalogosoknak, ahogy azt némelyik országban alkalmazzák. Természetesen beépíthetnénk a programba a Közép-Európában megszokott piros- és zöldfényű gyalogos jelzőlámpát is, csakhogy ez a tanulókészlet a közlekedési lámpában alkalmazott LED-eken kívül már csak egy további LED-et tartalmaz.

Építsen be a következő kísérlethez még egy LED-et az előtétellenállásával együtt a kapcsolási rajz szerint. Ezt a 24. számú GPIO-portra csatlakoztassa.



4.1 ábra 4.1: A dugasztábla beültetése a közlekedési lámpához és a gyalogos villogófényhez.

A szükséges alkatrészek: 1 db dugasztábla
1 db piros LED 1 db sárga LED
1 db zöld LED 1 db kék LED
4 db 220-ohmos ellenállás 5 db összekötőkábel



4.2 ábra 4.2: Közlekedési lámpa gyalogos villogófénnyel.

Az ampel02.py program vezérli az új jelzőlámpát. Az előző verzióhoz képest a program csekély mértékben kibővült.

```
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 ("Strg+C beendet das 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 Így működik

A program menete messzemenően ismert. A most egy kissé hosszabb piros fázis alatt a kék gyalogos jelzőlámpának gyorsan kell villognia.

blau = 4 Egy új változó definiálja a gyalogos jelzőlámpa LED-jét a listában.

Ampel=[4,18,23,24] A lista négy elemre bővült a négy LED vezérlési lehetősége miatt.

GPIO.setup(Ampel[blau], GPIO.OUT, initial=False) Megtörténik az új LED inicializálása, amely kezdetben ki van kapcsolva. Ez az alapbeállítás a közlekedési lámpa zöld fázisa alatt.

```
            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)
```

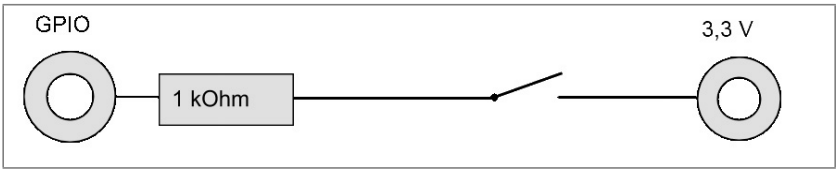
A jelzőlámpa ciklusban a piros fázis kezdete után 0,6 másodperccel elkezdődik egy hurok, amely villogtatja a kék LED-et. Erre a célra egy for-hurok szolgál, amely a korábbi kísérletekben alkalmazott while-hurokkal ellentétben mindig egy adott számú hurokmenetet alkalmaz, és nem fut mindaddig, amíg nem teljesül egy meghatározott megszakítási feltétel.

for i in range(10): Mindegyik for-huroknak egy hurokszámlálóra van szüksége, egy változóra, amely minden egyes hurokmenet után új értéket vesz fel. Az egyszerű hurokszámláló számára az összes programozási nyelvben az i változónév honosodott meg. Természetesen minden más név is megengedett. Ez az érték, mint bármely más változó a hurkon belül, lekérdezhető, de nincs rá szükség. A range paraméter a hurokban megadja azt, hogy hányszor fut le a hurok, pontosabban, hogy mily értékeket vehet fel a hurokszámláló. Példánkban a hurok tízszer fut le. Az i hurokszámláló eközben 0-tól9-ig terjedő értékeket kap. A hurkon belül az új kék LED bekapcsolódik, és 0,05 másodperc múlva újra kikapcsolódik. További 0,05 másodperc múlva befejeződik egy hurokmenet, és a következő megint a LED bekapcsolódásával indul. Ily módon tízet villan, ami összesen 1 másodpercig tart.

time.sleep(0.6) Az utolsó hurokmenet után 0,6 másodperc késleltetéssel folytatódik a közlekedési lámpa normális kapcsolási ciklusa, amennyiben a már világító piros LED mellé a sárga is bekapcsolódik. Eddig még nincs semmi új a nap alatt. Valóban érdekessé akkor válik a gyalogos jelzőlámpa, ha nem automatikusan fut, hanem egy gombnyomásra kell elindulnia, mint ahogy ez az eset sok gyalogoslámpa működésében. A következő kísérletben egy GPIO-portra csatlakoztatott nyomógomb fogja szimulálni egy igazi gyalogos jelzőlámpa nyomógombját.

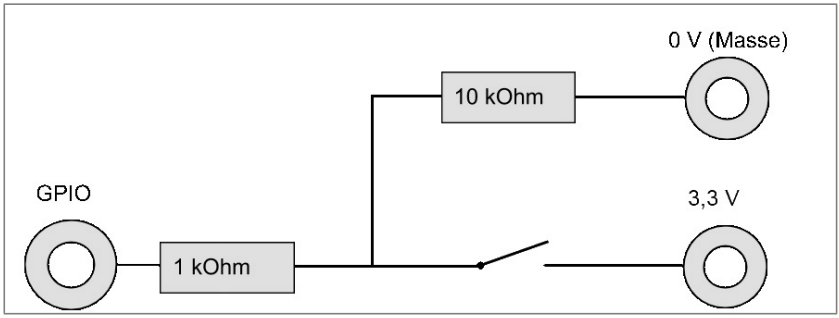
4.2 Nyomógomb a GPIO-porton

A GPIO-portok nemcsak adatokat tudnak kibocsátani, pl. LED-ek által, hanem adatbevitelre is alkalmazhatók. Ehhez a programban bemenetként kell definiálni őket. A beadásra a következő kísérletben egy közvetlenül a dugasztáblába bedugott nyomógombot használunk. A nyomógombnak négy csatlakozópontja van, amelyek közül kettő-kettő (nagy távolság) össze van kötve egymással. Amíg nyomva tartja a nyomógombot, mind a négy csatlakozópont össze van kötve egymással. Egy kapcsolóval ellentétben a nyomógomb nem reteszelődik. Az összeköttetés a gomb felengedésekor azonnal megszűnik. Ha az egyik bemenetként definiált GPIO-porton egy +3,3 V-os jel van, akkor annak a logikai értéke True (igaz), ill. 1 . Elméletileg tehát egy nyomógombbal az adott GPIO-port összeköthető a Raspberry Pi +3,3 V-os csatlakozójával, de semmi esetre se tegye ezt! A GPIO-port ettől túlterhelődne. Iktasson be minden esetben a GPIO-bemenet és a +3,3 V-os érintkező közé egy 1 kohm-os védőellenállást annak a megakadályozására, hogy túl nagy áram folyjon a GPIO-porton, és ezáltal a processzoron.



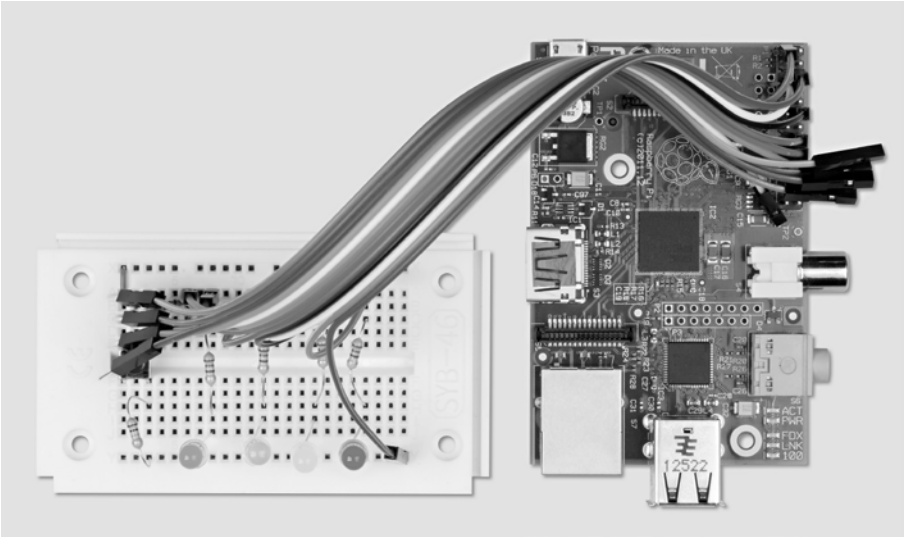
4.3 ábra 4.3: Nyomógomb védőellenállással egy GPIO-bemeneten.

A legtöbb esetben máris működne ez az egyszerű kapcsolás, de nyitott nyomógomb mellett nem lenne egyértelműen definiált a GPIO-port állapota. Ha egy program lekérdezi ezt a portot, véletlen eredményt kaphat. Ennek a megakadályozására egy viszonylag nagy ellenállást – általában 10 kohmot – kötünk a portról a testre. Ez az un. pull-down (lehúzó) ellenállás nyitott nyomógomb esetén ismét lehúzza a GPIO-port állapotát 0 V-ra. Mivel az ellenállás értéke nagyon nagy, amíg a nyomógombot nyomva tartjuk, nincs rövidzárveszély sem. A nyomógomb megnyomott állapotában a +3,3 V és a testvezeték közvetlenül össze van kötve ezen az ellenálláson keresztül.



4.4 ábra. 4.4: Nyomógomb védőellenállással és lehúzó ellenállással egy GPIO-bemeneten.

Építsen be az alábbi ábrának megfelelően egy nyomógombot a két ellenállással a kapcsolásba.

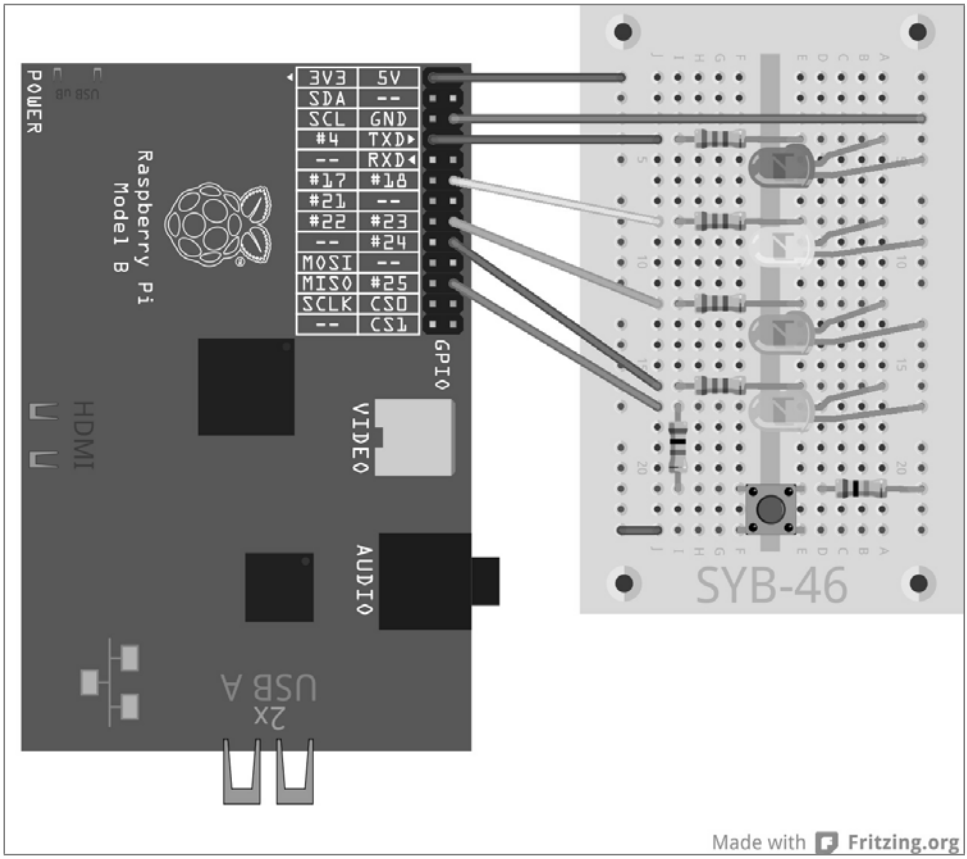


4.5 ábra 4.5: A dugasztábla beültetése a szükség szerint működő gyalogos jelzőlámpához

A szükséges
alkatrészek: 1 db
dugasztábla
1 db piros
LED 1 db
sárga LED
1 db zöld
LED 1 db
kék LED
4 db 220-ohmos
ellenállás 1 db
nyomógomb
1 db 1 kohmos ellenállás
1 db 10 kohmos
ellenállás 7 db
összekötőkábel
1 db rövid huzaláthidaló

A nyomógombnak az ábrán látható alsó érintkezőkapcsa a dugasztábla pozitív sínjén keresztül össze van kötve a Raspberry Pi +3,3 V-os vezetékével (1. csap). A nyomógombnak a pozitív sínnel való összekötésére a rajz áttekinthetősége érdekében egy rövid huzaláthidalót alkalmazunk. Másképp közvetlenül is összeköthetjük a nyomógomb alsó érintkezőjét egy összekötőkábel segítségével a Raspberry Pi 1. csapjával.

A nyomógombnak az ábrán látható felső érintkezőkapcsa egy 1 kohmos védőellenálláson (barna- fekete-piros) keresztül a 25. GPIO-porttal, és egy 10 kohmos lehúzó ellenálláson (barna- fekete-narancs) keresztül a testvezetékkel van összekötve.



4.6 ábra 4.6: Gyalogos villogófény nyomógombbal.

Az ampel03.py program vezérli az új nyomógombos gyalogos villogófény jelzőlámpa-elrendezést.

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

GPIO.setmode(GPIO.BCM)



rot     = 0; gelb    = 1; gruen  = 2; blau    = 3; gomb = 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[gomb],  GPIO.IN)

print ("Taster drücken, um
Fußgängerblinklicht einzuschalten, Strg+C
beendet das program")

try:
    while True:
        if
            GPIO.input(Ampel[gomb]
            )==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
    KeyboardInterr
    upt:
    GPIO.cleanup()
```

4.2.1 Így működik

A program a legutóbbi verzióhoz képest egy kissé ki lett egészítve.

-*- coding: utf-8 -*- Ahhoz, hogy a német umlautos betűk a Fußgängerblinklicht szó programkiírásában helyesen jelenjenek meg – függetlenül attól, hogyan van beállítva az IDLE-felület a felhasználónál –, az elején egy kódot definiálunk a különleges karakterek megjelenítése érdekében. Ennek benne kell lennie az összes olyan szövegeket kiíró programban, amelyekben umlautok vagy az adott nyelvre jellemző különleges karakterek is vannak.

ASCII, ANSI és Unicode

A normál (latin) ábécé 26 betűből (a magyar 40-ből) áll, és még néhány umlautos/ékezetes betűből, mind nagybetűs és kisbetűs írásmódban, ehhez járul még a tíz számjegy és néhány írásjel; mintegy 100 különböző karakter adódik ki ezekből. Egy byte segítségével 256 különböző karakter jeleníthető meg. Ennek tehát elegendőnek kellene lennie - így gondolták a számítógép történetének a kezdetén, amikor a mai technika legfontosabb alapelveit meghatározták.

Elég hamar kiderült, hogy a 256 karakterből álló ASCII (American Standard Code for Information Interchange = kb. amerikai szabvány információcsere kódok) karakterkészlet kitalálói tévedtek. Az amerikaiak nem láttak túl az angol nyelvterületen. Az összes fontos világnyelvben, a teljesen sajátos írást használó keletázsiai és arab nyelveket nem is számítva, több száz betűt kell megjeleníteni. Közülük csak kevés fért el a 256 karaktert átfogó lista üres helyein.

panaszkodnak, az angol megjegyzéseket magunk se értjük meg már valamikor később, a britek nevetnek a rossz angolságon.

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

Az egyszerűség kedvéért a nyomógomb számára a 4 számmal jelölt kiegészítő elemet és a 25 GPIO-portot iktattuk be a listába. Ezen az úton a nyomógomb számára könnyen választhatunk másik GPIO-portot is, mivel annak a száma a LED-ek GPIO-portjaival együtt csak ezen a helyen kerül be a programba.

GPIO.setup(Ampel[gomb], GPIO.IN) A nyomógomb GPIO-portja bemenetként kerül meghatározásra. Ez a meghatározás elvégezhető a GPIO.setup-on keresztül is, ezúttal azonban a GPIO.INparaméterrel történik.

```
print ("Taster drücken, um Fußgängerblinklicht
einzuschalten, Strg+C beendet das Programm")
```

Indításkor a program egy bővebb üzenetet küld, amely közli a felhasználóval, hogy nyomja meg a nyomógombot.

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

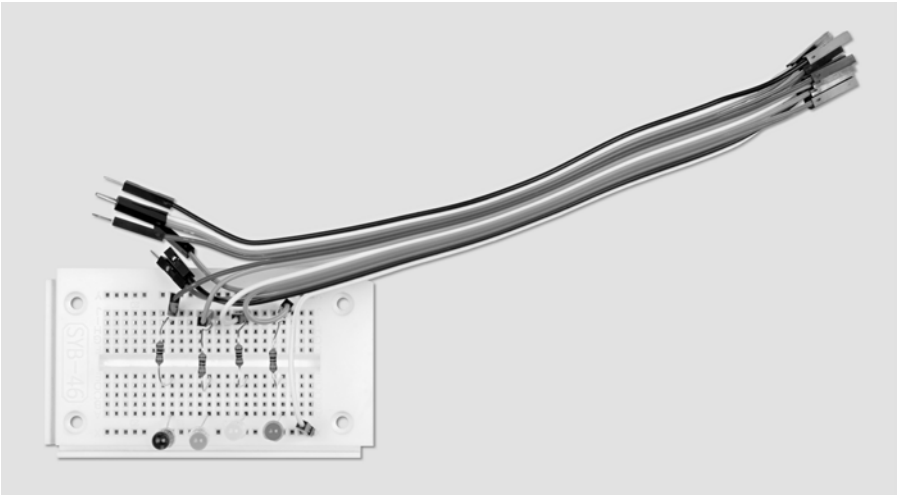
A végtelen hurokba most be van építve egy lekérdezés. A következő utasításokat csak akkor hajtja végre a program, ha a 25. GPIO-port felvette a True értéket, azaz a felhasználó megnyomta a nyomógombot. Eddig az időpontig a közlekedési lámpa a zöld fázison áll. A hurok további menete lényegében megfelel az utolsó programéval. A közlekedési lámpa a sárgán keresztül pirosra vált, a gyalogos villogófény tízet villan. Ezután a közlekedési jelzőlámpa a piros-sárgán keresztül zöldre vált.

time.sleep(2) Egy kis különbség van azonban ebben a programban. A 2 másodpercig tartó zöld fázis most a hurok végére van beiktatva, és nem a hurok elejére. Ennek ellenére hurokmenetenként egyszer alkalmazásra kerül, azzal a különbséggel, hogy a jelzőlámpa-ciklus azonnal, és nem késleltetéssel indul el, ha megnyomjuk a nyomógombot. Annak a megakadályozására, hogy a zöld fázis majdnem kimaradjon, ha a nyomógombot közvetlenül a sárga fázis után megint megnyomjuk, ez a késleltetés a hurok végére van beiktatva.

5 Tarka LED-minták és futófények

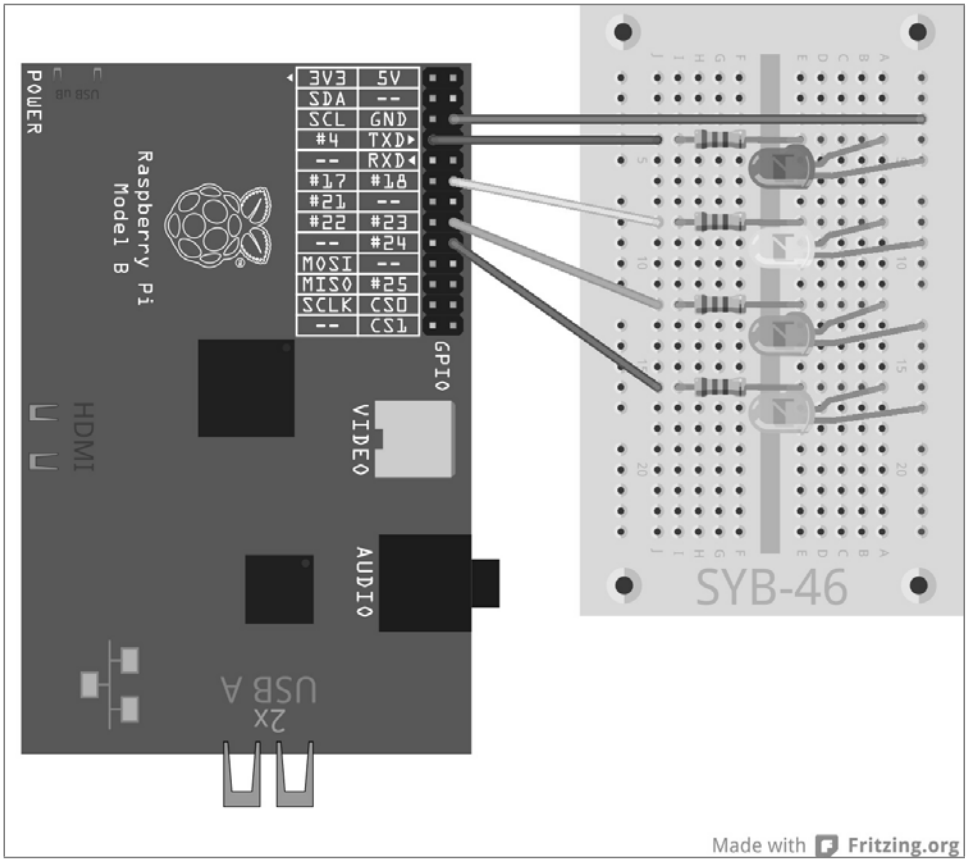
A futófény mindig kedvenc effektus volt a figyelem felhívására pincepartikon vagy profi fényreklámokon. A Raspberry Pi és pár LED segítségével valami ilyesmit is megvalósíthatunk.

Építsük meg a következő kísérletet négy LED-del és előtétellenállásaikkal az ábrának megfelelően. Ez a kapcsolás megfelel a nyomógomb nélküli gyalogos jelzőlámpának az előző kísérletből.



5.1 ábra 5.1: A dugasztábla beültetése a tarka fényminták és a futófények számára.

A szükséges alkatrészek: 1 db dugasztábla 1 db piros LED 1 db sárga LED 1 db zöld LED 1 db kék LED 4 db 220-ohmos ellenállás 5 db összekötőkábel



5.2 ábra 5.2: Négy LED előtétellenállásokkal.

Különböző LED-villogási minták alapján ismertetünk további hurkokat és programozási módszereket a Pythonban. A következő program különböző LED-mintákat ajánl, amelyek közül a felhasználó tasztatúra-beadással választhat.

A ledmuster.py program a LED-eket különféle fénymintákban villogtatja.

```
# -*- 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 ("Lichteffekte zur Auswahl"); print ("1 – Lauflicht zyklisch")
print ("2 – Lauflicht hin und zurück"); print ("3 – auf- und absteigend")
print ("4 – alle blinken gleichzeitig"); print ("5 – alle blinken zufällig")
print ("Strg+C beendet das program")

try:
    while True:
        e = raw_input ("Bitte muster auswählen: ")
        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 ("Ungültige Eingabe") ("Érvénytelen beadás")

except KeyboardInterrupt:
    GPIO.cleanup()
```

5.1.1 Így működik

A program első sorait az UTF-8-kódolás megadásával és a szükséges könyvtárak importálásával a korábbi kísérletekből már ismerjük. Kiegészítésül itt még a random könyvtár importálása történik meg egy véletlenszerű villogási minta előállításához.

Ennek a programnak a fejlesztésekor arra helyezték a hangsúlyt, hogy lehetőleg sokoldalúan legyen használható, azaz gond nélkül ki lehessen bővíteni négynél több LED-re. A jó programozási stílushoz ma már hozzátartozik az ilyen rugalmasság. A Raspberry Pi példáján az így programozott programok nem csak új GPIO-portokkal bővíthetők, hanem könnyen át is írhatók más GPIO-portokra, ha erre hardvertechnikailag szükség van.

LED = [4,18,23,24] A LED-ek számára ismét egy lista készül GPIO-számokkal, hogy ezek a portok csak egy helyen kerüljenek be a programba.

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

Ahelyett, hogy a LED-ek GPIO-portjait a korábbi programoknak megfelelően egyenként inicializálnánk, ezúttal egy for-hurok fut a LEDlistával. Az i hurokszámláló egyenként kiveszi az egyes értékeket a listából, példánkban a LED-ek GPIO-portszámait, és nem növeli csak úgy egyesével a számokat, mint az eleddig alkalmazott for-hurkokban. Ezzel a módszerrel tetszőleges hosszúságú listák dolgozhatók fel. A lista hosszúságát a program fejlesztésekor még csak nem is kell ismerni.

A LED-ek négy GPIO-portját kimenetként határozzuk meg, és értéküket 0 -ra állítjuk, hogy lekapcsolódjanak a korábbi kísérletek után égve maradt LED-ek.

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

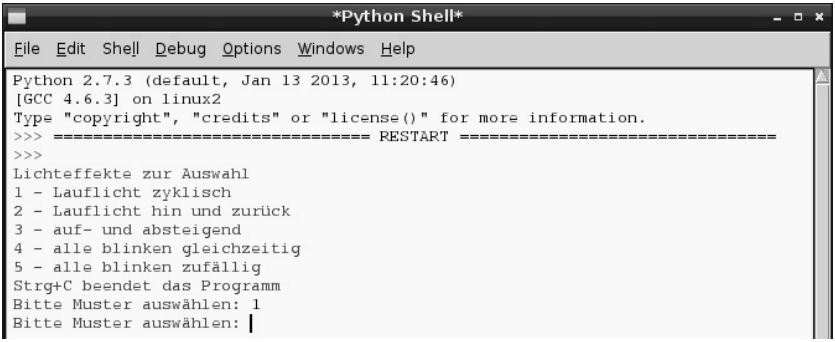
Ahhoz, hogy a program végérvényesen és könnyen változtatható maradjon, még három változót definiálunk:

z	A LED-ek száma	A LED-ek számát a program a len() funkció segítségével automatikusan átveszi a LED listából.
[W]	Ismétlések	Mindegyik fényminta a jobb felismerhetőség érdekében alapvetően ötször ismétlődik. Ez a szám tetszés szerint megváltoztatható, és utána mindegyik mintára érvényes.
t	Idő	Ez a változó azt adja meg, hogy mennyi ideig legyen bekapcsolva a szünet ugyanennyi ideig tart. A t név azokra a változókra, amelyek időtartamot tárolnak a programokban, majdnem az összes meghonosodott.

A változókként definiált értékek csak ezen a helyen kerülnek be a programba, és könnyen megváltoztathatók. Ezek után a meghatározások után indul a tulajdonképpeni program.

```
print ("Lichteffekte zur Auswahl"); print ("1 – Lauflicht zyklisch")
print ("2 – Lauflicht hin und zurück"); print ("3 – auf- und absteigend")
("Kiválasztható fényeffektusok".....("1 - ciklikus futófény")...("2 -
```

Ezek a sorok egy útmutatót jelenítenek meg a képernyőn a felhasználó számára arról, hogy melyik számjegygombbal melyik fénymintát hozhatja létre.



5.3 ábra 5.3: A program a képernyőn.

Miután megjelent a választék a képernyőn, elindul a program főhurka. Ehhez itt is a while True:-végtelenhurkot alkalmazzuk, amely egy try...except-utasításba van beágyazva.

e = raw_input ("Bitte minta auswählen: ") ("Válassza ki a mintát:") Rögtön a hurok elején a program egy beadást vár a felhasználó részéről, amely az e változóban kerül tárolásra. A raw_input() funkció normál szövegként átveszi a beadást feldolgozás nélkül. Ezzel ellentétben azinput() funkcióba beadott

matematikai műveleteket vagy változóneveket közvetlenül feldolgozza a program. A legtöbb esetben tehát a raw_input()
a jobb választás, mivel nem kell tépelődnie a lehetséges beadások esetlegességével.

A program vár, amíg a felhasználó egy betűt bead, majd megnyomja az [Enter]-nyomógombot. Attól függően, hogy melyik számot adta be a felhasználó, elő kell jönnie egy meghatározott LED-mintának. Ennek a lekérdezésére egy if...elif...else-szerkezetet alkalmazunk.

1. minta
Ha a beadás egy 1 volt, akkor az ezen sor utáni beugrasztott programrész végrehajtást nyer.

if e == "1": Vegye figyelembe, hogy a beugrasztásnak a Pythonban nem csak optikája van. Ugyanúgy, mint a hurkokat, egy ilyen lekérdezést is egy beugrasztás vezet be.

Az egyenlő nem mindjárt azonos
A Python kétféle egyenlőségi jelet használ. Az egyszerű = arra szolgál, hogy egy változóhoz meghatározott értéket rendeljen hozzá. A kettőzött egyenlőségi jelet lekérdezésekben használja, és azt vizsgálja vele, hogy két érték tényleg egyenlő-e egymással.

Ha tehát a felhasználó egy 1 -et ad be a tasztatúrával, elindul egy hurok, amely egy ciklikus futófényt állít elő. Ez a hurok az összes alkalmazott LED-mintára elvileg azonos felépítésű.

for i in range(w): A külső hurok annyiszor ismétli meg a fénymintát, ahogy az az elején definiált w változóban meg lett adva. Ezen a hurkon belül van egy másik, amely a mindenkori mintát előállítja. Ez a hurok mindegyik mintára más.

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

Az egyszerű ciklikus futófény esetében ez a hurok a listában szereplő minden egyes LED-re egymás után végigfut. Hogy hány LED-ről is van szó, az a program elején a z változóban van tárolva. A hurokszámláló aktuális állásának megfelelő sorszámú LED bekapcsolódik. Ezután a program az elején a t változóban tárolt ideig várakozik, majd azután ismét kikapcsolja a LED-et. Majd elkezdődik a következő hurokmenet a következő LED-del. A külső hurok a teljes belső hurkot ötször megismétli.

2. minta
Ha a felhasználó egy 2-est adott be, egy hasonló hurok kezdődik el. Itt a LED-eket nem csak az egyik irányban számlálja le a program, hanem a futófény végén ismét, de az ellenkező sorrendben. A fény váltakozva fut előre és vissza.

elif e == "2": A további lekérdezések az első után az elif lekérdezést alkalmazzák, ami azt jelenti, hogy csak akkor kerülnek végrehajtásra, ha az előző lekérdezés eredménye a False (hamis) volt.

```
            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)
```

Itt is egymásba skatulyázott hurkokat alkalmazunk. Az előzőleg ismertetett programrésznek megfelelő első belső hurok után, azaz miután kigyulladt a 3-es számú LED, elkezdődik egy további hurok az ellenkező irányú futófény számára. A lista elemei minden esetben 0-val kezdődő számozásúak. A negyedik LED ezek után a 3-es számot viseli.

Egy hurok visszafelé történő futtatásához a for...range() kibővített szintaxist alkalmazzuk. A csak egy végérték megadásával ellentétben három paramétert is megadhatunk: a kezdeti értéket, a lépés nagyságát és a végértéket. Példánkban ezek a következők:

Kezdeti érték	z-1	A z változó tartalmazza a LED-ek számát. Mivel a listák elemeinek a számozása 0-val kezdődik, az utolsó LED sorszáma z-1.
Lépésméret	-1	Ha a lépésméret -1 , akkor minden egyes hurokmenet egy számmal visszaszámlál.
Végérték	-1	Egy hurokban a végérték mindig az az első érték, amelyet nem érünk el. Az első előreszámláló hurokban a hurokszámláló 0-val kezd, és példánkban a 0, 1, 2, 3értéket veszi fel a LED-ek címzéséül. A négyszeri hurokmenet esetén az érték nem éri el a 4 -et. A visszaszámláló huroknak a 0 -val kell végződnie, és így elsőként a -1 az az érték, amelyet nem érhet el.

A már ismertetett második hurok a négy LED-et sorjában a fordított irányban gyújtja ki. Ezután a külső hurok újraindítja a teljes ciklust, amely itt, mivel mindegyik LED kétszer villog, kétszer annyi ideig tart, mint az első programrészben.

3. minta

Ha a felhasználó egy 3-ast adott be, egy hasonló hurok kezdődik el. A LED-eket itt is mindkét irányban megszámlálja a hurok, de nem kapcsolódik ki azonnal a bekapcsolás után.

```
        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)
```

Az első belső hurok a LED-eket egymás után késleltetéssel kapcsolja be. A hurok végén, amely a time.sleep(2*t) sor kiugrasztásáról ismerhető fel, a késleltetési idő dupláját várja ki a hurok. Ez idő alatt az összes LED világít. Ezután elkezdődik egy újabb hurok, amely visszaszámlál, és az egyik LED-et a másik után kapcsolja ki. Itt is a hurok végén, amikor már az összes LED kialudt, a késleltetési idő dupláját várja ki a hurok, mielőtt a külső hurok még egyszer elindítaná a teljes ciklust.

4. minta

Ha a felhasználó egy 4-est ad be, egy másik villogási minta kezdődik el, amelynél az összes LED egyszerre villog, és nem hívja le őket egymás után a program.

```
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)
```

Miután nem kapcsolható több GPIO-port egyszerre be- és ki egyetlen utasítással, itt is hurkokat alkalmazunk, de a hurkon belül késleltetés nélkül. A négy LED közvetlenül egymás után kapcsolódik be. Az emberi szem számára ez egyidejűnek látszik. Az első belső hurok végén a program kivárja a kétszeres késleltetési időt, mielőtt az összes LED-et újra kikapcsolná.

A különböző világítási és sötétség-idő miatt különféle fényeffektusok jönnek létre villogófényben. A villogás jobban észrevehető, ha a világítási idő hosszabb, mint a sötét állapot ideje. A nagyon rövid világítási idő viszonylag hosszú sötét állapot mellett villanófény effektust kelt.

5. minta

Ha a felhasználó egy 5-öst adott be, a LED-ek teljesen véletlenszerűen villognak.

```
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)
```

Mivel itt nem alkalmazunk egymásba skatulyázott hurkokat, a hurkok többször futnak le. A w és a z változó többszörözése által mindegyik LED átlagosan annyiszor villog, mint az első mintában.

A random.randint() funkció egy véletlenszámot ír be a j változóba. Ez a véletlenszám az első paraméternél nagyobb vagy egyenlő, míg a második paraméternél kisebb vagy egyenlő, a példánkban tehát a 0, 1, 2, 3 értékeket veheti fel.

A véletlenül kiválasztott LED bekapcsolódik, majd a késleltetési idő után ismét kikapcsolódik. Ezután újraindul a hurok, és egy új LED választódik ki véletlenszerűen.

Érvénytelen beadás

Az összes, a felhasználó részéről beadást kérő program esetében a hibás beadásokat el kell csípni. Ha a felhasználó valami nem várt dolgot ad be, a programnak reagálnia kell rá.

```
else
    print ("Ungültige beadás") ("Érvénytelen beadás")
```

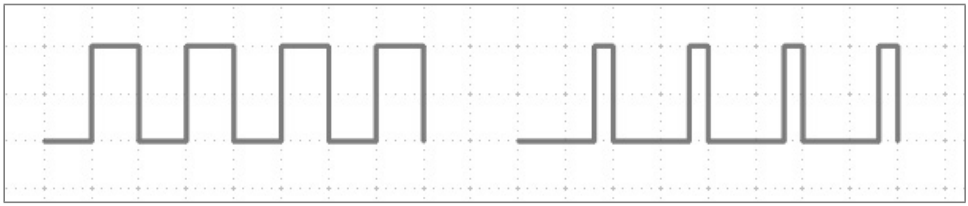
Ha a felhasználó valami mást adott be, az else alatt beadott utasítás végrehajtódik. Ez a lekérdezést tartalmazó szakasz mindig akkor igaz, ha más lekérdezések nem adnak igaz eredményt. Esetünkben a program egy üzenetet jelenít meg a képernyőn.

Mint a korábbi kísérletekben is, a programot egy KeyboardInterrupt fejezheti be, amennyiben a felhasználó megnyomja a [Strg]+[C] nyomógomb-kombinációt. Az utolsó sor lezárja a használt GPIO-portokat, és ezáltal az összes LED kialszik.

6 A LED-ek fényszabályzása impulzusszélesség-modulációval

A LED-ek tipikusan jelkiadásra szogáló alkatrészek a digitális elektronikában. Két különböző állapotot tudnak felvenni, ezek a be és a ki, 0 és 1, vagy True (igaz) és False (hamis). Ugyanez érvényes a digitális kimenetként definiált GPIO-portokra. Ennélfogva elméletileg nem volna lehetséges a LED-ek fényszabályzása.

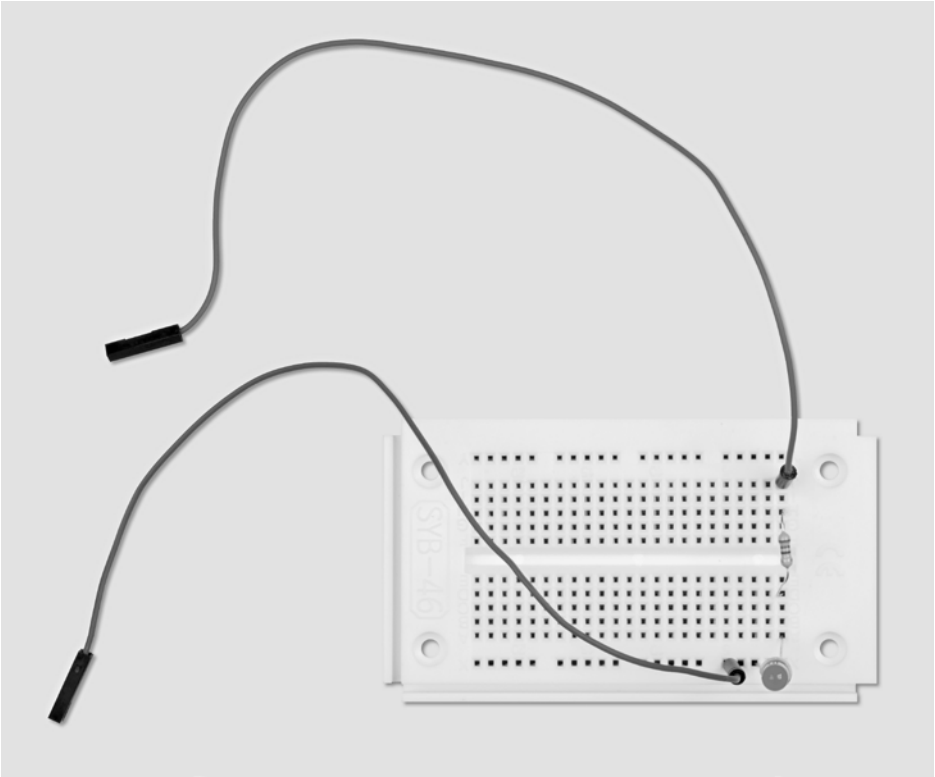
Egy trükkel azonban mégis csak lehet egy digitális GPIO-portra kötött LED fényességét szabályozni. Ha elég gyorsan villog egy LED, az emberi szem azt már nem észleli villogásként. Az impulzusszélesség modulációnak nevezett technika egy pulzáló jelet állít elő, amely nagyon rövid időközökben bekapcsolódik és kikapcsolódik. A jel feszültsége mindig azonos marad, csak a False szint (0 V) és a True szint (+3,3 V) közötti viszony változik. Az impulzuskitöltési tényező a bekapcsolt állapot időtartamának és egy kapcsolási ciklus teljes időtartamának az aránya.



6.1 ábra 6.1: Balra: kitöltési tényező 50 % – jobbra: kitöltési tényező 20 %.

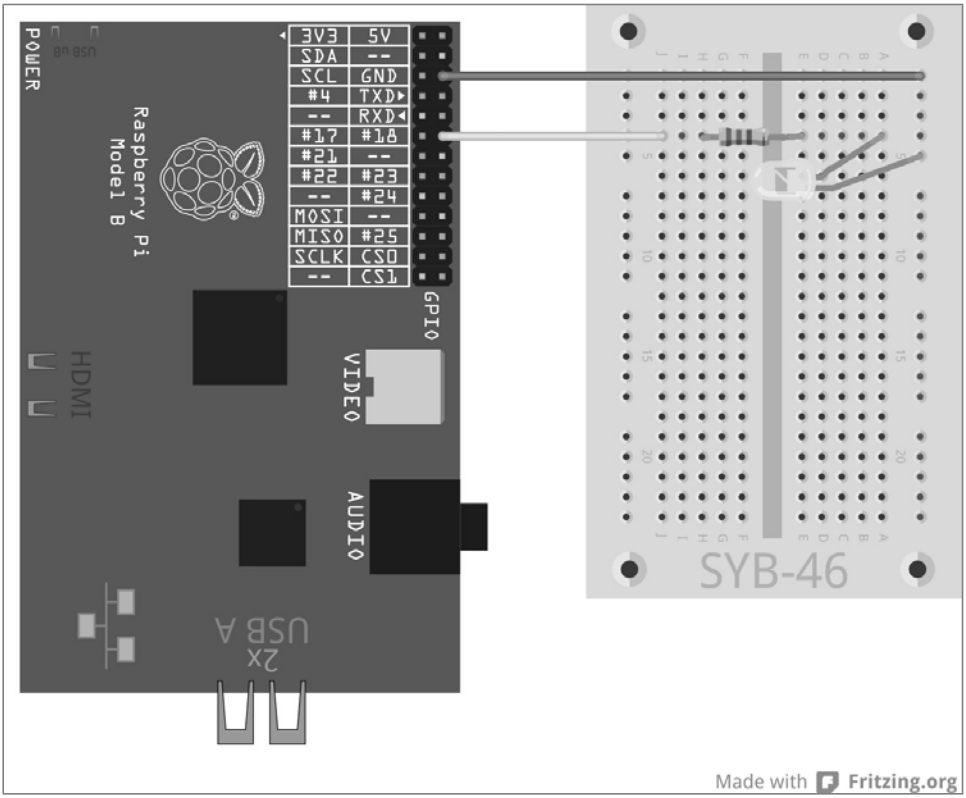
Minél kisebb a kitöltési tényező, annál rövidebb a LED világítási ideje egy kapcsolási cikluson belül. Emiatt a LED sötétebbnek hat, mint egy állandóan világító LED.

Csatlakoztasson a következő kísérlethez egy LED-et egy előtétellenálláson keresztül a 18. GPIO-portra.



6.2 ábra 6.2: A dugasztábla beültetése egy LED-del.

A szükséges alkatrészek: 1 db dugasztábla
1 db sárga LED
1 db 220-ohmos ellenállás 2 db összekötőkábel



6.3 ábra 6.3: Egy LED a 18. GPIO-porton.

A leddimmen01.py program a LED fényét ciklikusan szabályozza fényesebbre és sötétebbre, és ehhez a GPIO-könyvtár saját PWM-funkcióját alkalmazza. A PWM-jel saját thread-ként lesz generálva. Ezen a módon egy fényszabályzott LED (szinte) úgy használható egy programban, mint egy normálisan világító LED.

```
import RPi.GPIO as GPIO
import  time
GPIO.setmode(GPIO.BCM);  LED  =  18
GPIO.setup(LED,    GPIO.OUT)
print  ("Strg+C  beendet  das  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 Ez így működik:

A program egy része ismerősnek tűnhet, néhány eleme azonban nem, mert ezen a helyen kiruccanunk az objektum orientált programozáshoz. A kezdésnél, mint ismert, könyvtárakat importálunk. Ezúttal csak egyetlen LEDváltozót határozunk meg a kimenetként inicializált 18. GPIO-port számára.

print ("Strg+C beendet das program") ("A Ctrl + C befejezi a programot")Mivel ez a program is atry...except-szerkezettel fut, és a felhasználónak kell leállítania, erről megfelelő tájékoztatást kap a felhasználó.

p = GPIO.PWM(LED, 50) A GPIO-könyvtárból vett GPIO.PWM() funkció döntő fontosságú a PWM-jelek kiadása szempontjából. Ennek a funkciónak két paraméterre van szüksége, a GPIO-portra és a PWM-jel frekvenciájára. Esetünkben a GPIO-portot a LED változó határozza meg, a frekvencia (másodpercekénti rezgésszám) 50 Hz .

Miért az 50 Hz az ideális frekvencia a PWM számára?
Az emberi szem a 20 Hz-nél gyorsabb fényváltozásokat már nem észleli. Mivel Európában a váltóáramú hálózat frekvenciája 50 Hz, sok fénycső ezzel a frekvenciával villog, amelyet a szemünk nem vesz észre. Ha egy LED 20 Hz-nél nagyobb, de 50 Hz-nél kisebb frekvenciával villogna, interferencia jöhetne létre más fényforrásokkal, ami miatt a fényszabályozás hatása nem érvényesülne egyenletesen.

A GPIO.PWM() egy úgynevezett objektumot hoz létre, amely a p változóban lesz tárolva. Az ilyen objektum sokkal több, mint csak egy egyszerű változó. Az objektumoknak különféle tulajdonságaik lehetnek, és úgynevezett metódusokkal lehet befolyásolni őket. A metódusok, egy ponttal elválasztva, közvetlenül az objektumnév mögött vannak megadva.

p.start(0) A start() metódus elindítja a PWM-jel generálását. Ehhez még meg kell adni egy impulzuskitöltési tényezőt. Esetünkben a kitöltési tényező 0, a LED tehát mindig ki van kapcsolva. Most elindul a végtelen hurok, amelybe közvetlenül be van ágyazva két egymás utáni hurok, amelyek váltakozva fényesebbé és sötétebbé teszik a LED-et.

for c in range(0, 101, 2): A hurok 2-es lépésekben számol 0-tól 100-ig. Egy for-hurok végéül mindig azt az értéket kell megadni, amelyet éppen nem érünk el, esetünkben a 101-et.

p.ChangeDutyCycle(c) Mindegyik hurokmenetben a ChangeDutyCycle() metódus a PWM-objektum kitöltési tényezőjét a hurokszámláló értékére állítja, tehát minden alkalommal 2%-kal magasabbra, amíg csak az utolsó menetnél nem áll 100%-on , és a LED a teljes fényerővel világít.

time.sleep(0.1) Mindegyik hurokmenetben 0,1 másodperc várakozási idő után nő meg 2%-kal a kitöltési tényező a következő menetre.

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

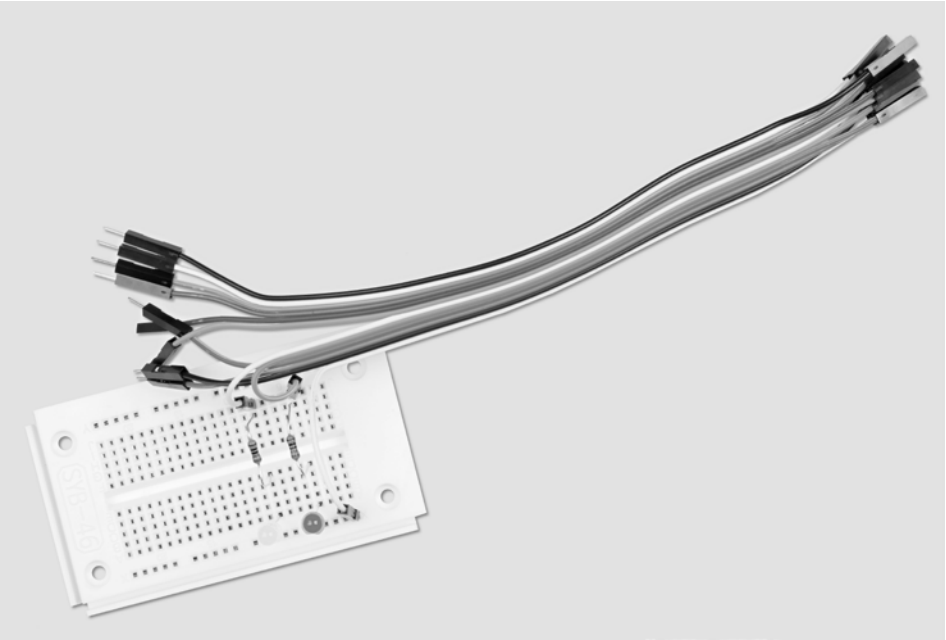
Miután a LED elérte teljes fényességét, egy második hurok azonos séma szerint visszafelé számlál. Ez a hurok 100-tól -2-es lépésekben lefelé számlál. Ez a ciklus addig ismétlődik, amíg a felhasználó meg nem nyomja a [Strg]+[C] nyomógomb-kombinációt.

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

A KeyboardInterrupt (nyomógombos megszakítás) kiváltja most kiegészítőleg a PWM-objektum stop() metódusát. Ez a metódus befejezi a PWM-jel előállítását. Ezután, mint az utóbbi programokból már ismerjük, a GPIO-portok visszaállítása megy végbe.

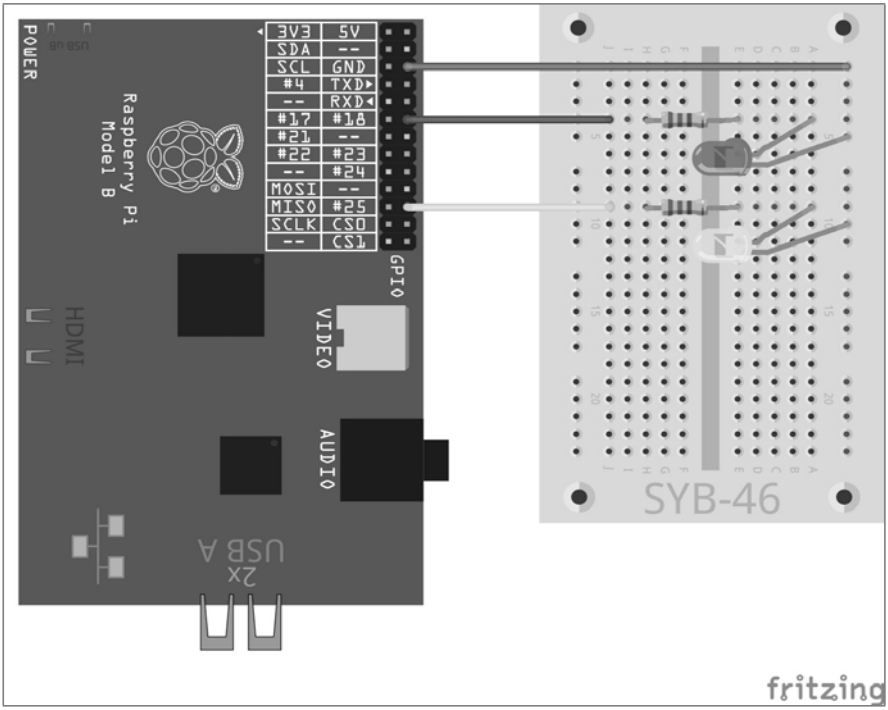
6.1.2 Két LED egymástól független fényszabályzása

Mivel a PWM-jel programozásához a Python-szkriptben nincs szükség programidőre, több LED fényességét is lehet egymástól függetlenül szabályozni, amint azt a következő kísérlet bemutatja. Csatlakoztasson ehhez egy további LED-et egy előtétellenálláson keresztül a 25. GPIO-portra.



6.4 ábra 6.4: A dugasztábla beültetése két LED fényszabályzásához.

A szükséges
alkatrészek: 1 db
dugasztábla
1 db sárga
LED 1 db
piros LED
2 db 220-ohmos
ellenállás 3 db
összekötőkábel



6.5 ábra 6.5: Egy második LED a 25. GPIO-porton

A leddimmen02.py program ciklikusan szabályozza az egyik LED fényét világosabbra és sötétebbre, mialatt a másik LED fényét ugyan az elsővel együtt világosabbra szabályozza, a másik ciklusban azonban a LED nem válik sötétebbé, hanem ismét 0-ról világosodik, és közben gyorsan lobog a fénye.

```
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 ("Strg+C beendet das 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 Ez így működik:

A program alapszerkezete megfelel az előző kísérletével, egy pár kiegészítéssel.

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

A GPIO-port számára elhelyezett változó helyett most két változóból álló lista van definiálva, és így két GPIO-port, a 18. és a 25., van kimenetként inicializálva a két LED számára.

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

Ezután a két objektum, a p és a q is elhelyezésre kerül, amelyek előállítják a PWM-jelet a két LED számára, mindkettőhöz 50 Hz frekvenciával.

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

Az első hurokban a két PWM-objektum kitöltési tényezője egyidejűleg növekszik lépésről lépésre. Ebben a fázisban a két LED azonos módon viselkedik.

q.ChangeFrequency(10) Ennek a huroknak a végén, amikor mindkét LED elérte a teljes fényességét, a második LED PWM-jelének a frekvenciáját 10 Hz-re csökkenti a ChangeFrequency() metódus. Ezt a frekvenciát a szemünk még villogásnak érzékeli.

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

            time.sleep(0.1)
```

Most elindul a második hurok, az áttekinthetőség kedvéért ezúttal növekvő számlálással. A pPWM-objektumból való, az ebben a ciklusban lépésről lépésre leszabályozandó fényű első LED számára a kitöltési tényező megfelelő értékei minden egyes menetben kiszámítódnak. A q PWM-objektumból való második LED kitöltési tényezője egyszerűen ismét felfelé számlálódik. A villogó hatást a megváltoztatott frekvencia hozza létre.

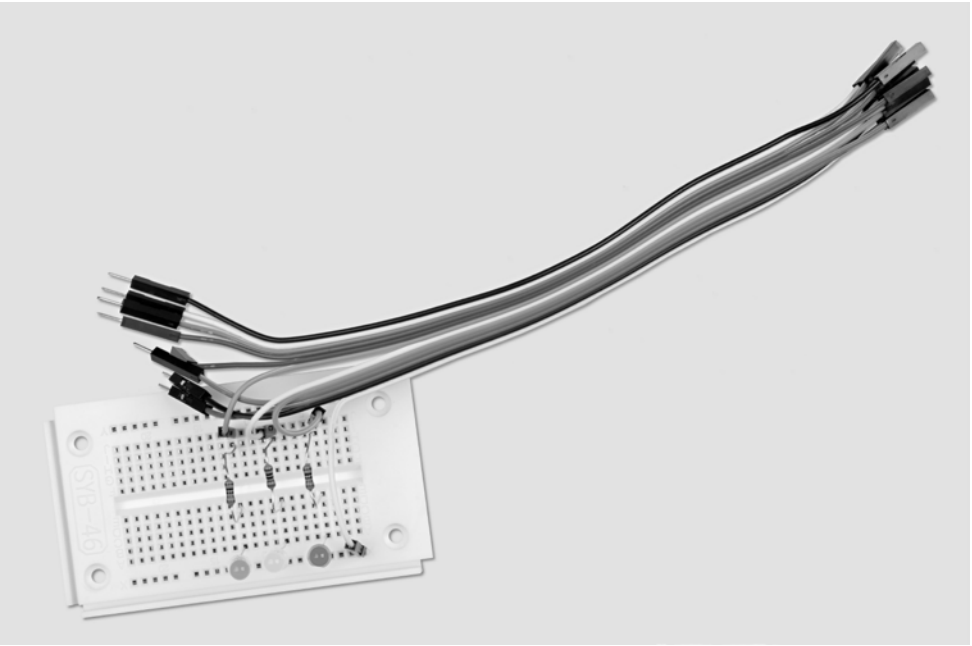
q.ChangeFrequency(50) A második hurok végén ennek a LED-nek a frekvenciája visszaállítódik 50 Hz-re, hogy a következő ciklusban megint pontosan úgy, mint az első LED, lassan váljon fényesebbé.

7 Memóriakártya töltöttségi állapotának a kijelzése LED-ekkel

A memóriakártyák a merevlemezekhez hasonlóan mindig nagyon gyorsan megtelnek. Pedig szeretnénk, ha lenne egy egyszerű optikai töltöttségi állapot kijelzőnk, hogy egy pillantással felmérhessük, mikor van már fogytán a tárolóhelyünk. Három LED segítségével ez a feladat kényelmesen megoldható a Raspberry Pi-on. Ehhez az operációs rendszernek a Python által lekérdezett funkcióit használjuk.

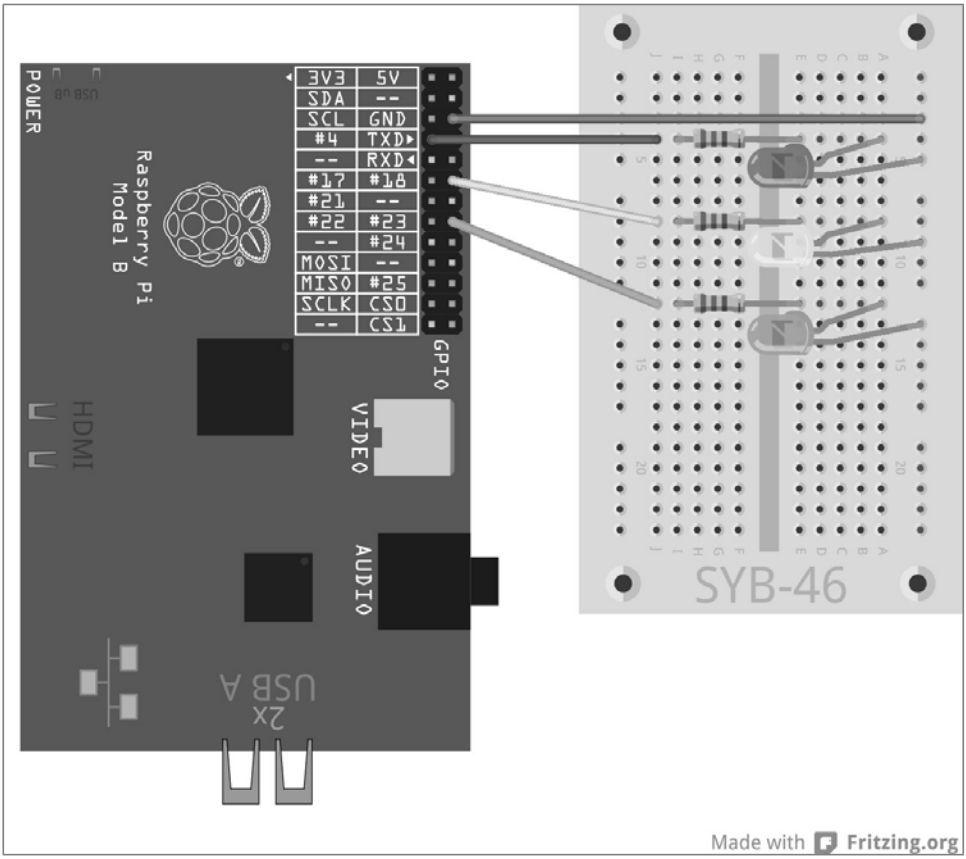
7.1 ábra 7.1: Természetesen a szabad tárolóhelyet közvetlenül is megjeleníthetjük a Raspberry Pi fájlkezelőjében.

A szabad tárolóhely megjelenítésére a jelzőlámpa-kapcsolás három különböző színű LED-jét használjuk fel.



7.2 ábra 7.2: A dugasztábla beültetése a memóriakártya töltöttségi állapotának a kijelzésére.

A szükséges
alkatrészek: 1 db
dugasztábla
1 db piros
LED 1 db
sárga LED
1 db zöld
LED
3 db 220-ohmos
ellenállás 4 db
összekötőkábel



41. 7.3: Három LED jelzi a memóriakártyán lévő szabad tárolóhelyet.

A speicheranzeige.py program a szabad tárolóhelytől függően különféle LED-jelzést szolgáltat:

szabad memóriahely	LED-jelzés
< 1 MB	piros
1 MB-tól 10 MB-ig	piros–sárga
10 MB-tól 100 MB-ig	sárga
100 MB-tól 500 MB-ig	sárga–zöld
> 500 MB	zöld

7.1 táblázat 7.1: Ilyen a memóriakártya töltöttségi állapotának a kijelzése

```
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 program")

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()
```

Futtassa a programot, a LED-ek folyamatosan jelzik a memóriakártyán lévő szabad tárolóhelyet. Próbálja ki a programot az által, hogy nagy fájlokat másol a hálózaton keresztül a memóriakártyára, majd letörli őket. A kijelzés automatikusan frissül.

7.1.1 Így működik

A program az os Python-modult alkalmazza a szabad tárolóhely kiszámítására, amely az alapvető operációs rendszer funkciókat bocsátja rendelkezésre.

import os Az os modult, mint más modulokat is, a program ellején kell importálni.

g1 = 1; g2 = 10; g3 = 100; g4 = 500 Ezek a sorok a szabad tárolóhely tartományhatárait definiálják, ahol a kijelzésnek át kell kapcsolnia. A program az egyszerűség kedvéért a megabájtot és nem a bájtot alkalmazza, mivel ezeket a számokat jobban el tudjuk képzelni. A határokat bármikor másképp állíthatjuk be, csupán a négy értéket kell nagyság szerint elrendezni.

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

Egy lista definiálja a három LED GPIO-portjainak a számát. Majd egy hurok a három GPIO-portot kimenetként inicializálja, és az összes LED-et kikapcsolt állapotba állítja.

Ebben a kísérletben is egy try...except-szerkezetet és egy végtelenhurkot alkalmazunk azért, hogy a program automatikusan újra és újra fusson mindaddig, amíg a felhasználó meg nem szakítja a [Strg]+[C] gombkombinációval. Ezután következnek az igazán érdekes funkciók, amelyek hozzányúlnak az operációs rendszerhez, és lekérdezik a szabad tárolóhelyet.

s = os.statvfs('/') Az os-könyvtárból való os.statvfs() statisztikai modul különféle statisztikai információkat szállít a fájlrendszer számára, amelyeket a végtelenhurkon belül minden egyes hurokmenetben újólag objektumként be kell írni az s változóba.

f = s.f_bsize * s.f_bavail / 1048576 Most az s.f_bsize metódus egy memóriablokk méretét szolgáltatja bájtban. Az s.f_bavail a szabad blokkok számát adja meg. A két érték szorzata adja ki ennélfogva a szabad bájtok számát, amelyet itt 1.048.576-tal elosztva kapjuk meg a szabad megabájtok számát. Az eredmény az f változóban lesz tárolva.

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

Ha a szabad tárolóhely kisebb, mint az első határérték (1 MB), akkor a bekapcsolt LED-ek fénymintáját megadó xkaraktersor "100" -ra állítódik. Az első, a piros LED-nek világítania kell. A minta egy egyszerű karaktersorozat, amely a 0 és az 1 számból áll.

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

Az elif-lekérdezések segítségével a további határértékek lekérdezése is végbemegy, és annak megfelelően a LED-minta beállítása is, ha az első lekérdezés nem igaz, azaz több mint 1 MB szabad tárolóhely áll rendelkezésre.

```
    else:

      x = "001"
```

Ha egyik lekérdezés se volna igaz, azaz a felső határérték által megadottnál több szabad tárolóhely áll rendelkezésre, a LED-minta "001"-re állítódik. Az utolsó, zöld LED-nek kell világítania.

```
    for i in range(3):

      GPIO.output(LED[i], int(x[i]))
```

Egy hurok határozza meg a három LED GPIO-kiviteli értékeit. Az összes LED sorra megkapja a karaktersor mindenkori sorszámának megfelelő számértéket, a 0-át vagy az 1-et. A 0 és az 1 érték ugyanúgy használható, mint a False és a True , a GPIO-kimenetek ki- vagy bekapcsolására. Az int() funkció megállapítja egy karakterből annak a számértékét. A karaktert az i hurokszámláló olvassa ki a minta karaktersorozatának az adott helyéről.

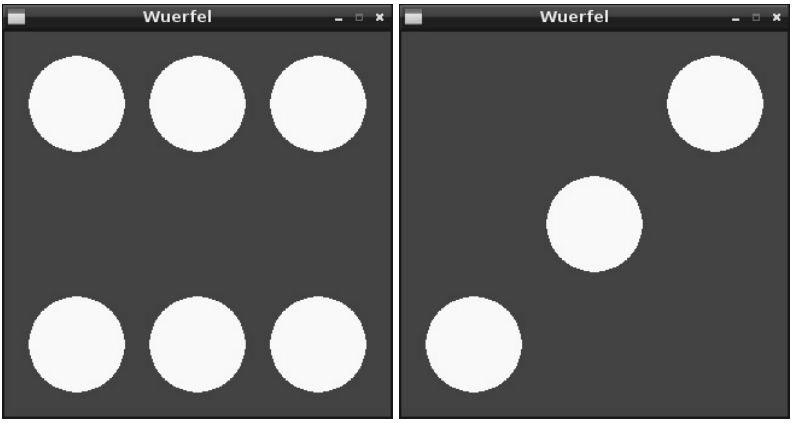
time.sleep(1.0) A program 1 másodpercig vár a következő hurokmenetig. A teljesítmény-megtakarítás céljából hosszabb várakozási időt is beállíthat a szabad tárolóhely megállapításának a megismétléséig.

Ezen a helyen újra kezdődik a while...True-hurok. Ha a felhasználó időközben megnyomta a [Strg]+[C] nyomógomb-kombinációt, kiváltódik egy KeyboardInterrupt , és elhagyjuk a hurkot. Ezután lezárulnak a GPIO-portok, és ezzel kikapcsolódnak a LED-ek.

8 Grafikus dobókocka

Egy korszerű játékhoz grafika is kell, nemcsak szövegkiírás, mint a legelső DOS-számítógépek idején. A PyGame könyvtár gyárilag programozott funkciókkal és objektumokkal szolgál a grafikus megjelenítéshez és a játékprogramozáshoz. Ezáltal nem kell mindent az alapoktól kezdve újra kitalálnunk.

Sok játékhoz van szükségünk dobókockára, de gyakran nincs kéznél. A következő példaprogram bemutatja, hogy milyen egyszerűen lehet dobókockaként használni a Raspberry Pi-t a Python és a PyGame segítségével:



8.1 ábra 8.1: A Raspberry Pi mint dobókocka.

A dobókockának lehetőleg egyszerűnek, és csak egy nyomógombbal kezelhetőnek kell lennie, és a kockadobás véletlen eredményének grafikusan, mintegy "valódi" dobókockán kell megjelennie. A következő, wuerfel.py nevű program egy ilyen dobókockát szimulál a képernyőn.

```
# -*- coding: utf-8 -*-
import pygame, sys, random
```

A "sudo" nélkül fut
Mivel ennek a programnak nincs szüksége a GPIO-portokra, korlátozás nélküli felhasználói felhatalmazás nélkül is futtatható. A Python-IDE egyszerűen az asztalon lévő IDLE szimbólummal indítható el.

8.1.1 Így működik

Ez a program számos új funkciót mutat be, különösen a PyGame könyvtár segítségével történő grafikus megjelenítéshez, amelyet természetesen nemcsak játékokhoz, hanem a képernyőn megjelenítendő bármilyen más grafikához is alkalmazni lehet .

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

Ez a három programsor majdnem minden, a PyGame modulokat alkalmazó program elején ott áll. A már említett, véletlenszámokat generáló random modulon kívül maga a pygame modul, továbbá a sys modul is betöltésre kerül, mivel ez fontos, a PyGame számára szükséges rendszerfunkciókat tartalmaz, például az ablakok megnyitását és bezárását. A PyGame-könyvtár összes funkciója importálásra kerül, majd a tulajdonképpeni PyGame-modul inicializálására kerül sor.

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

Ez a fontos funkció minden, a grafikus megjelenítést használó programban egy karakterfelületet, az úgynevezett Surface-t, definiál, amely példánkban 320 x 320 pixel méretű, és a FELD nevet kapja. Vegye észre a kettős zárójeles írásmódot, amelyet elvileg a grafikus képernyő koordinátáihoz használunk. Egy ilyen Surface a képernyő egy új ablakában jelenik meg.

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

Ez a sor viszi be az ablak nevét (Wuerfel = dobókocka).

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

Ezek a sorok definiálják a két használt színt, a kéket és a fehéret. Minden alkalommal közvetlenül is beadhatnánk a színek értékét a programba, ami azonban éppen nem növeli meg az áttekinthetőséget.

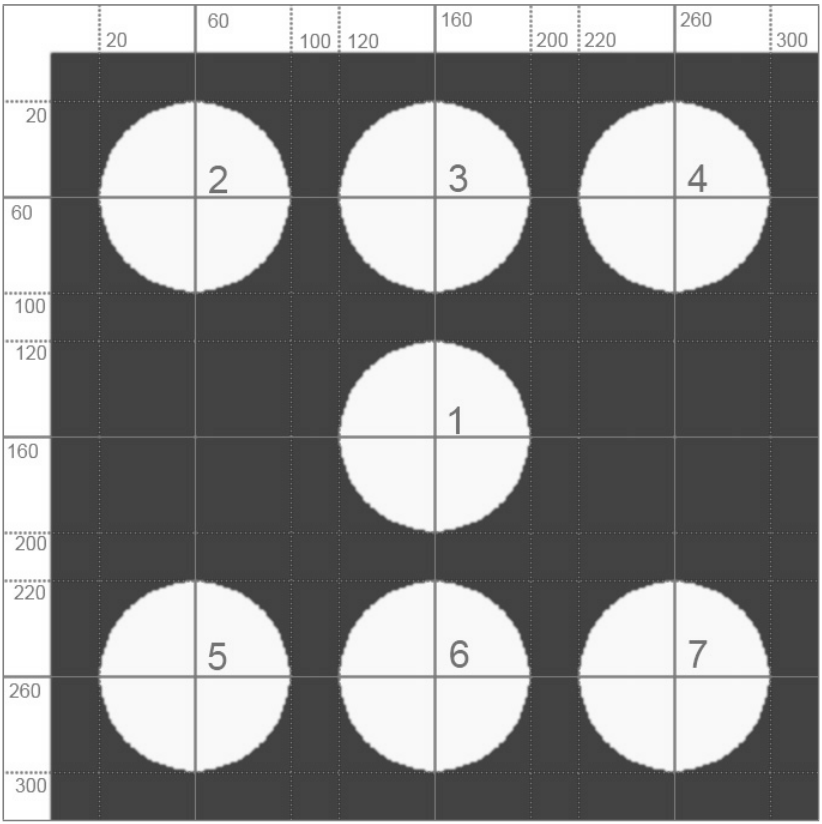
A színek megjelenítése a képernyőn
A színeket a Python, mint a legtöbb más programozási nyelv is, 0 és 255 közé eső három számmal definiálja, amelyek a piros, a zöld és a kék

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

Ezek a sorok határozzák meg a kocka pöttyeinek a középpontját. A 320 x 320 pixel méretű karaktermezőben a kocka pöttyeinek az egyes tengelyei a 60, 160 és 260koordinátákon fekszenek .

A számítógépes grafika koordinátarendszere
Egy ablak, ill. egy Surface-objektum minden egyes pontját egy x- és egy y-koordináta jelöli meg. A koordinátarendszer nullapontja az iskolában tanultaktól eltérően nem balra lent, hanem balra fent van. Ugyanúgy, mint

A hét pont, a P1 - P7 jelölik a kocka pöttyeinek a grafikában megadott középpontjait. A kocka mindegyik pöttyének a sugara 40 pixel. 80 pixel tengelytávolság mellett a kocka pöttyei, ill. a pöttyök és az ablak szélei között 20 pixel távolság marad.



8. 2 ábra 8.2: A kocka pöttyei és azok koordinátái.

Ezen a helyen a többi változóval együtt még egy mainloop nevű változó állítódik a True értékre, amelyre később a játék főhurkában lesz szükség.

mainloop = True Ezzel elintéztük az alap-tennivalókat, és elkezdhetjük a tulajdonképpeni játékot.

"Beliebige Taste drücken, um zu würfeln, [Esc]

Ez a sor röviden elmondja a felhasználónak, hogy mit kell tennie. A tasztatúra bármelyik gombjának a megnyomására egy új kockadobás történik. A print mindig a Python-Shell-ablakba ír ki, és nem az új grafikus ablakba.

while mainloop: Most kezdődik el a játék főhurka. Sok játék végtelen hurkot alkalmaz, amely ismételten és állandóan valamilyen aktivitásra szólítja fel a felhasználót. Valahol a hurokban definiálva kell lennie egy megszakítási feltételnek, amely arról gondoskodik, hogy be lehessen fejezni a játékot.

Erre a célra alkalmazzuk itt a mainloop változót, amely csak a két Boole-féle értéket, a True és a False (igaz és hamis, be és ki) értéket veheti fel. Ez a változó kezdetben a True értéken áll, és minden hurokátmenetnél lekérdezésre kerül. Ha a hurok folyamán a False értéket veszi fel, a hurok a következő menet előtt befejeződik.

for event in pygame.event.get(): Ez a sor kiolvassa a legutolsó felhasználói aktivitást, és event-ként (esemény) tárolja. A játékban csak kétfajta játékra irányuló felhasználói aktivitás van: A felhasználó megnyom egy gombot, és ezáltal dob, vagy a felhasználó be akarja fejezni a játékot.

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

A játék befejezésére két lehetőség van: Rákattinthat az ablak jobb felső sarkában látható x-szimbólumra, vagy megnyomhatja az [Esc] gombot. Ha rákattint az x-szimbólumra, akkor az operációs rendszer által szolgáltatott eseménytípus: event.type == QUIT. Ha megnyom egy gombot, majd felengedi, az eseménytípus event.type == KEYUP. Kiegészítésül még ebben az esetben a megnyomott gomb tárolódik is az event.key helyen.

Az ismertetett if-lekérdezés megvizsgálja, hogy a felhasználó be akarja-e zárni az ablakot, vagy (or) megnyomott és felengedett egy nyomógombot, és (and) ennek a nyomógombnak a belső megjelölése a K_ESCAPE-e. Ha ez az eset, akkor a mainloop változó a False értékre állítódik, ami a játék főhurkát a következő menet előtt befejezi.

if event.típuse == KEYDOWN: A felhasználói aktivitás másik fajtája az, ami a játék közben ismét és ismét, és nemcsak egyszer fordul elő, hogy a felhasználó megnyom egy nyomógombot. Nem játszik szerepet ebben, hogy az [Esc]-gombon kívül melyiket nyomja meg. Mihelyt megnyom egy gombot (KEYDOWN), elindul egy fontos programrész, amely a kockadobás eredményét állítja elő, és meg is jeleníti azt.

FELD.fill(BLAU) Elsőként a FELD megnevezésű Surface-objektum, a tulajdonképpeni programablak a kezdéskor kéknek BLAU definiált színnel töltődik ki, hogy átfesse az előző kockadobás eredményét.

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

Most a random véletlen-funkció egy 1 és 6 közé eső véletlenszámot generál, és elmenti a ZAHL változóba.

print ZAHL Ez a sor csak ellenőrzésül kiírja a kockadobás eredményét a Python-Shell-ablakba. Ezeket a sorokat el is hagyhatja, ha lemond a szövegalapú megjelenítésről.

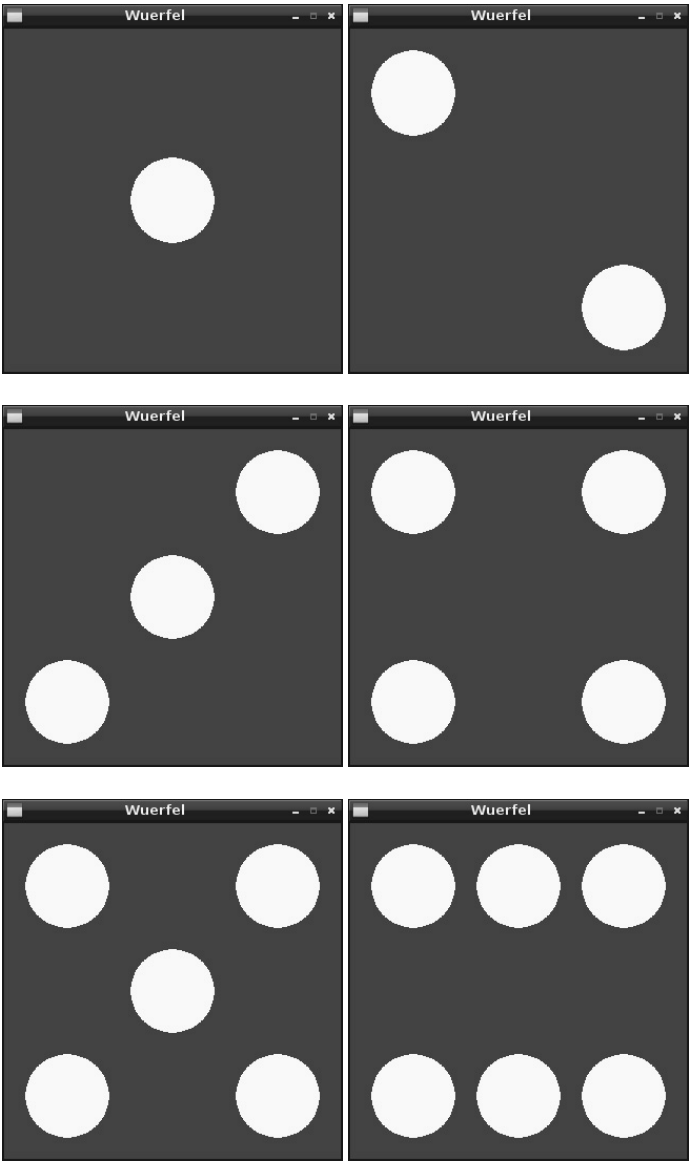
```
            if ZAHL == 1:
                pygame.draw.circle(FELD, WEISS, P1, 40)
```

Most ugyanazon séma szerint következik hat lekérdezés. Ha a véletlen kockadobás egy adott értékű számot adott, annak megfelelően egytől hatig terjedő pöttyöt jelenít meg. Az erre alkalmazott pygame.draw.circle() funkció négy vagy öt paramétert igényel:

- A *Surface* azt a karakterfelületet adja meg, amelyre a megjelenítés történik, például a FELD felületet.
- A *Farbe* adja meg a kör színét, példánkban az előbb definiált WEISS (fehér) színt.
- A *Mittelpunkt* adja meg a kör középpontját.
- A *Radius* adja meg a kör sugarát.
- A *Dicke* adja meg a körvonal vastagságát. Ha elhagyja ezt a paramétert, vagy 0 -ra állítja, a kör kitöltődik.

Ha az if-feltételek közül teljesül egy is, a kocka pöttyei egyelőre csak egy virtuális karakterfelületen tárolódnak.

pygame.display.update() Csak ez, a hurok végén lévő sor aktualizálja a grafikát a képernyőn. Most tényleg láthatók a kocka pöttyei.



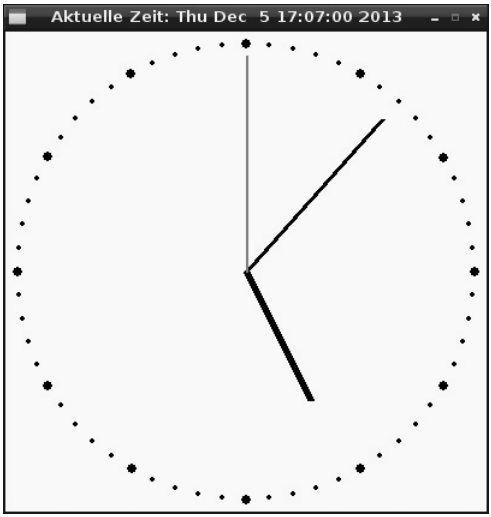
8.3 ábra 8.3: A kockadobás lehetséges hat eredménye.

A hurok azonnal újraindul, és ismét a felhasználó egy gombnyomására vár. Ha a hurok folyamán a mainloop False értékre lett állítva, mert a felhasználó be akarja fejezni a játékot, a hurok nem fog még egyszer lefutni, hanem ehelyett a következő sor kerül végrehajtásra:

apygame.quit() Ez a sor befejezi a PyGame-modult, ami által a grafikus ablak is bezárul, majd az egész program is.

9 Analóg óra a képernyőn

A digitális időkijelzés, amelyet ma már megszoktunk a számítógépektől, csak a 70-es években jött divatba. Azelőtt évszázadokig a pontos időt analóg jelezték ki egy számlapon mutatókkal. A digitális óra divatja az utóbbi években egy kicsit visszaesett, mivel felismerték, hogy az analóg órák gyorsabban, és rossz időjárási viszonyok között nagyobb távolságból is tisztán leolvashatók. Az emberi szem gyorsabban fog fel egy grafikát, mint egy számot vagy betűt. Az analóg óra jobban bevéső dik a rövid idejű memóriába, úgyhogy még akkor is, ha nem láttuk teljesen, vagy csak elmosódottan látjuk, mégis helyesen értelmezzük. Ha viszont pontatlanul látunk egy digitális órát, nem tudunk belőle megbízhatóan következtetni a kijelzett időre.



9.1 ábra 9.1: Analóg óra, PyGame programozással.

Ez a program nemcsak egy óra programozását akarja bemutatni, hanem az analóg megjelenítés alapelveit is, amelyeket nemcsak óráknál, hanem különböző mérési értékek vagy statisztikai adatok kijelzéséhez is alkalmazni lehet.

A kerek számlap középpontjában három óramutató jár, amelyek az órát, a percet és a másodpercet mutatják. Fent az ablak címhelyén fut még egy digitális időkijelzés is.

A uhr01.py program jeleníti meg az ábrán látható analóg órát a képernyőn:

```
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 Így működik

Ez a program a PyGame-könyvtár további funkcióit és a time-könyvtárat mutatja be, továbbá egyszerű trigonometriai szögfüggvényeket, amelyeket analóg megjelenítésekhez akalmazhatunk.

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

Az elején az előző programhoz hasonlóan a PyGame könyvtár inicializálása megy végbe. Ezenkívül importáljuk az időmeghatározásra szolgáló time-könyvtárat, valamint három funkciót a terjedelmes math-könyvtárból.

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

A grafikában alkalmazott három színt három változóban tároljuk.

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

Megnyílik egy 400 x 400 pixel méretű ablak, és teljesen kitöltődik fehér színnel.

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

Három változó rögzíti a középpontok koordinátáit, amelyekhez viszonyítva van az összes többi grafikus elem, a számlap és az óramutató. Az MX és MY változó tartalmazza a középpontok x- és y-koordinátáját, az MP változó a középpontot egy pontként, ahogyan a grafikus funkciókhoz alkalmazzuk.

Következőül egy fontos funkció definíciója jön, amely a középponttól való távolság és egy szög függvényében kiszámítja koordinátarendszerben lévő pontokat. Ez a funkció a programban többször lehívásra kerül mind a számlap, mind az óramutató megjelenítéséhez.

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

A funkció két paramétert alkalmaz: A a kívánt pontnak a középpontól vett távolsága, W a középponthoz viszonyított szög. Annak érdekében, hogy az óra megjelenítése egyszerűbb legyen, az óramutató szögét az óramutató járásával megegyező irányban a 12-órán átmenő függőleges vonalhoz mérten vesszük fel. A szöget sem fokban, hanem percben, a teljes kör 1/60 részeként, adjuk át a funkciónak. Az ilyen felvétellel sok köztes számítást takarítunk meg.

A Python a legtöbb programozási nyelvhez hasonlóan ívmérték szögegységben számol, és nem fokszámban. A math-könyvtár radian() funkciója átszámítja fokszámban vett szöget ívmértékben vett szögre. Ehhez a programban használt perc-szögadatot 6-tal megszorozza, hogy fokszám-adatot kapjon, majd 90 fokot levon belőle, hogy a 0-irány felfelé mutasson, mint a 0 perc minden egyes órában. Ez az ívmértékre átszámított szög a funkción belüli további számításokhoz a w1 változóban tárolódik.

Egy analóg óra megjelenítése a sinus és a cosinus szögfüggvényen alapszik . Általuk számítja ki a funkció egy pontnak a középponthoz (w1)viszonyított, ívmértékben vett szögéből koordinátáit a derékszögű koordinátarendszerben (x1 és y1). A középpont koordinátáit az MX és MY változókból veszi át, amelyek a funkción kívül lettek definiálva, és általános érvényűek. A pontnak a középponttól való távolságát az A paraméterből kapja a funkció. Az int() funkció állapítja meg az eredményből az egészrészt, mivel a pixelkoordináták csak egészrészként adhatók meg.

A funkció visszatérési értéke egy geometriai pont a kiszámított x1 és y1koordinátákkal, amelyek az összes többi ponttal megegyező módon kettős zárójelbe kerülnek.

Ennek a funkciónak a definiálása után a számlap megrajzolása jön.

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

Egy hurok rajzolja be egymás után a 60 percpontot egy körre. Az összes pontot a punkt()funkcióval határozza meg. A pontok azonos távolságra vannak a középponttól, és negyedenként 190 pixelnyire lévén még pontosan 10 pixel távolságra vannak az ablak szélétől. A pontok sugara 2 pixel.

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

Egy második hurok 12 nagyobb kört rajzol, amelyek az órákat jelölik a számlapon. Ezeknek a sugara 4 pixel, ezeket a hurok egyszerűen a meglévő körök fölé rajzolja, és teljesen lefedik azokat. Az órák szimbólumai egymástól öt percegységnyire lévő szögtávolságra vannak, amelyet a funkciónak átadott szögadatnak 5 -tel történő megszorzása ad ki.

mainloop = True; s1 = 0 Mielőtt elindul a program főhurka, még két segédváltozó kerül definiálásra, amelyekre a későbbiek folyamán lesz szükség. A mainloop adja meg a legutóbbi programpéldához hasonlóan azt, hogy a huroknak tovább kell-e futnia, vagy a felhasználó be akarja-e fejezni a programot. Az s1 tárolja az utoljára kijelzett másodpercet.

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

Most indul a program főhurka, amely mindegyik menetben, függetlenül attól, hogy milyen hosszú ideig tart, beírja az aktuális időt a zeit objektumba. Ehhez a time-könyvtárból való time.localtime() funkciót alkalmazza. Az eredmény egy adatstruktúra, amely különböző egyedi értékekből áll.

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

Az óramutató számára lényeges három érték, a másodperc, a perc és az óra, a strukturából három változóba íródik be, ezek az s, m ésh .

```
    if h > 12:
        h = h – 12
```

Az analóg órák csak tizenkét órát jeleznek ki. A time.localtime() funkció az összes időadatot 24-órás formátumban adja meg. A délutáni időadatból tehát egyszerűen 12 órát levon.

Időjelzés az analóg óráknál
Az alkalmazott mechanikától függően két különböző kijelzési módja van az analóg óráknak. A valódi analóg módon járó órák esetében a percmutató egyenletesen jár körbe, míg a digitálisan vezérelt órák, mint például a pályaudvari órák a teljes percnél ugranak egy percet tovább. Az utóbbi módszer előnye, hogy a pontos idő percre pontosan jobban leolvasható. A perc törtrészei a mindennapokban általában nem fontosak. Az óraprogramunkhoz mi is ezt a módszert választjuk. Az óramutatónak azonban egyenletesen kell körbejárnia. Itt nagyon furcsa és áttekinthetetlen lenne, ha minden teljes órában ugrana egy órával előre az óramutató.

hm = (h + m / 60.0) * 5 A hm változó percegységben tárolja az óramutató szögét, ahogy azt a teljes programban alkalmazza. Ezenkívül az aktuális órához hozzáadódik a percérték 1/60

része. Minden percben az óramutató egy óra 1/60 részével továbbmegy. A számított értéket 5 -tel megszorozza a program, mivel az óramutató óránként öt percegységet halad előre a számlapon.

if s1 <> s: Egy hurokmenet időtartama a programban nem ismert. Az analóg óra esetében ez azt jelenti, hogy a grafikát nem kell miden egyes hurokmenetben frissíteni, hanem csak akkor, ha az aktuális másodperc más mint az előzőleg kiírt. Ehhez később a programban a kiírt másodperc az s1 változóban áll.

Ha a másodperc megváltozott az utoljára kiírthoz képest, a most következő utasításokkal az óra grafikája frissül. Ha nem változott, akkor nem kell frissíteni a grafikát, és a hurok újraindul az aktuális rendszeridő lekérdezésével.

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

Elsőként egy fehér körfelületet rajzol a program, amely teljesen eltakarja az óramutatót. A 182 pixel méretű sugár egy kicsit nagyobb, mint a leghosszabbik mutató, hogy ne maradjon ott egy kis maradéka sem. Egy teljes felületű kört sokkal egyszerűbb rajzolni, mint a legutoljára kirajzolt mutató pixelenként pontosan átfesteni.

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

Ez a sor megrajzolja az óramutatót egy 6 pixel szélességű, a középponttól vett 120 pixel hosszúságú vonalként, a hm változó által megadott szögben. A pygame.draw.line() funkciót eddig még nem alkalmaztuk. Öt paraméterre van szüksége:

- A *Surface* adja meg azt a karakterfelületet, amelyre a rajzolás történik, ez lehet pl. a FELD.
- A *Farbe* adja meg a kör színét, példánkban az ellőbb definiált SCHWARTZ(fekete) színt.
- Az *Anfangspunkt* adja meg a vonal kezdőpontját, példánkban az óra középpontját.
- Az *Endpunkt* adja meg a vonal végpontját, példánkban ezt a punkt() funkció az óramutató szögéből számítja ki.
- A *Dicke* adja meg a vonal vastagságát.

Ugyanez a funkció rajzolja meg az óra két másik mutatóját is.

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

Ez a sor megrajzolja a percmutatót egy 4 pixel szélességű, a középponttól vett 170 pixel hosszúságú vonalként, a percérték által megadott szögben.

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

Ez a sor megrajzolja a másodpercmutatót egy 2 pixel szélességű, a középponttól vett 180 pixel hosszúságú piros vonalként, a másodpercérték által megadott szögben.

s1 = s Most az éppen kijelzett másodperc tárolásra kerül az s1 változóban, hogy ezt az értéket a következő hurokmenetekben össze lehessen hasonlítani az aktuális másodperccel.

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

Ez a sor digitális alakban kiírja az aktuális pontos időt az ablak címsorába. Ehhez a program a time.asctime() funkciót alkalmazza a time-könyvtárból, amely az időadatot készre formatált karaktersorozatként szállítja.

pygame.display.update() Eddig az összes grafikus elem csak virtuálisan került megrajzolásra. Csak ez a sor építi fel tényleg újra a grafikus megjelenítést. Egyidejűleg a frissítés is megtörténik. Emiatt semmiféle villódzás nem látható az egyes mutatók egymást követő felrajzolásakor.

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

Még mindig az if-lekérdezésen belül, tehát csak másodpercenként egyszer történik meg az esetleges rendszeresemények viszonylag teljesítményigényes lekérdezése, amellyel itt megállapítást nyer, hogy a felhasználó az utolsó másodpercen belül be akarta-e zárni az óraablakot, vagy megnyomta-e az [Esc]-gombot. Ha ez történt, a mainloop változó a False értéket kapja meg, és ennek következtében a hurok nem kapcsolódik be még egyszer.

pygame.quit() Az utosó sor először is bezárja a PyGame-modult, ami aztán a grafikus ablakot is bezárja, majd az egész programot is.

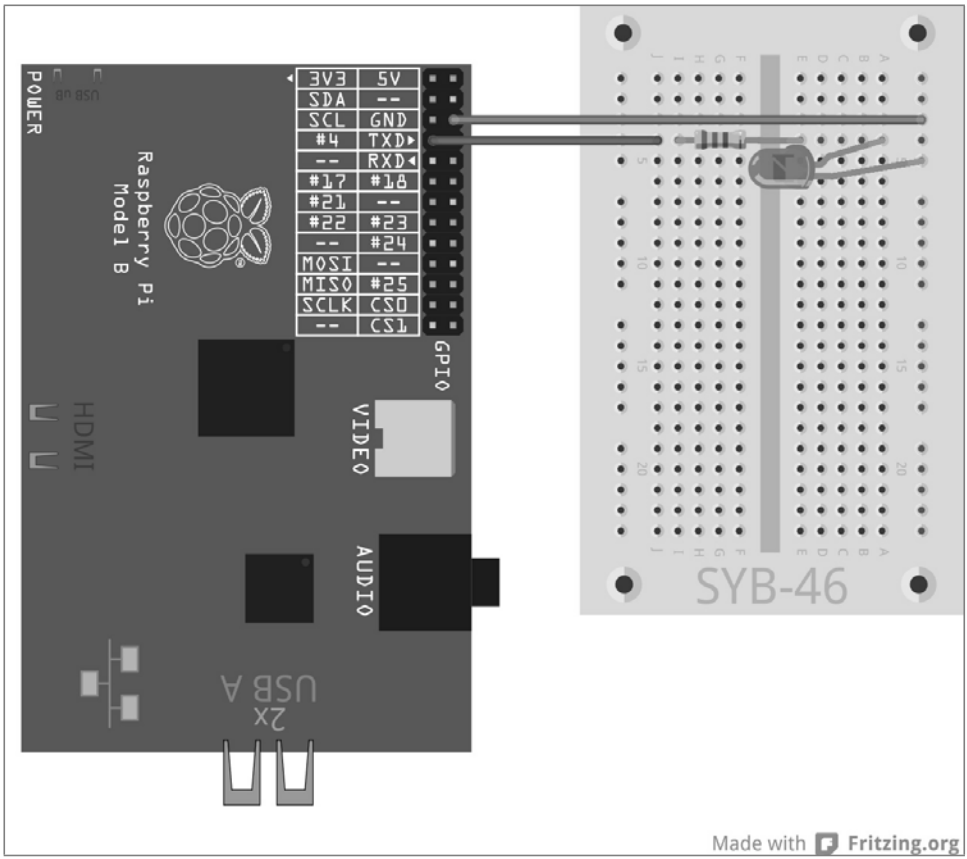
10 Grafikus párbeszédablak a programvezérléshez

Nincs olyan, a felhasználótól interakciót követelő modern program, amely tiszta szöveges üzemmódban futna. Mindenhol vannak olyan grafikus felületek, amelyeken nyomógombokra lehet rákattintani a tasztatúrán történő beadás kényszere helyett.

A Python maga nem nyújt grafikus felületeket a programok számára, de van számos külső modul, hasonlók a már ismertetett PyGame-hez, amelyek éppen grafikus felületek előállítására valók. Az egyik legismertebb ilyen modul a *Tkinter*, amely elérhetővé teszi a Python számára a *Tk*grafikus felületet, amelyet számos más programozási nyelv is használhat.

A Tk grafikus eszköztár struktúrái kissé eltérnek a Pythonétól, és első pillantásra szokatlannak is tűnhetnek. Ezért egy nagyon egyszerű példával kezdjük: Egy LED-et kell egy párbeszédablakban nyomógombokkal bekapcsolni és kikapcsolni.

A szükséges alkatrészek: 1 db dugasztábla
1 db piros LED
1 db 220-ohmos ellenállás 2 db
összekötőkábel



10.1 ábra 10.1: Egyetlen LED a 4. GPIO-porton.

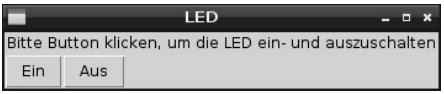
Csatlakoztasson egy LED-et egy előtétellenálláson keresztül a 4. GPIO-portra. A ledtk01.py program kigyújtja a LED-et.

```
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="Bitte Button klicken, um die LED ein- und
auszuschalten").pack()
Button(root, text="Ein", command=LedEin).pack(side=LEFT)
Button(root, text="Aus", command=LedAus).pack(side=LEFT)
root.mainloop()
GPIO.cleanup()
```



10.2 ábra 10.2: Így fog kinézni a kész dialógusmező.

10.1.1 Így működik

Ez a program bemutatja a Tkinter-könyvtár alapfunkcióit egy grafikus párbeszédablak elkészítéséhez. A PyGame grafika-könyvtárral ellentétben, amellyel a grafikák pixel pontossággal rajzolhatók fel, a Tkinterben a párbeszédablakok és a vezérlőelemek mérete automatikusan adódik ki, szükség esetén azonban utólag kézi úton módosítható.

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

A GPIO-könyvtár importálása után még a Tkinter-könyvtár elemei is importálásra kerülnek.

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

Ezek a sorok semmi újat nem mutatnak. A 4. GPIO-port kimeneti portként van definiálva egy LED számára, és LED változóként van megjelölve.

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

Most a LED bekapcsolását végző LedEin() funkciót definiáljuk.

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

Egy hasonló funkció, a LedAus(), kapcsolja ki megint a LED-et. Ezeket a funkciókat később a párbeszédmezőben lévő nyomógombokkal lehet felhívni.

Eleddig minden tiszta Python volt, most belevágunk a Tk-ba és tulajdonságaiba.

root = Tk() Tkinter úgynevezett widget-ekkel dolgozik. Itt önálló képernyőelemekről, a legtöbb esetben különféle elemeket tartalmazó párbeszédablakokról van szó. Mindegyik programnak egy root (gyökér) widgetre (eszköztár) van szüksége, amelyből kiindulva az összes további objektum lehívható. Ennek a root-widget-nek a neve mindigTk(), amely automatikusan generál egy ablakot, és inicializálja a Tkinter-könyvtárat is.

A Tkinterben lévő root.title("LED") objektumok különböző célokra szolgáló különféle módszereket bocsátanak rendelkezésre. Az egyik widgetben található title() metódus elhelyezi az ablak címét, kiírja tehát ebben az esetben a LED szót az új ablak címsorába.

Minden egyes widget több objektumot tartalmazhat, amelyeket egyenként kell definiálni. A Tkinter ismer ráadásul különféle objektumtípusokat, amelyekből mindegyik különféle paraméterrel meg lehet írni az objektum tulajdonságait. A paraméterek vesszőkkel elválasztva, az objektumtípus mögött zárójelbe téve vannak megadva. Mivel ez a lista nagyon hosszúvá válhat, rendszerint minden egyes paramétert külön sorba írunk, úgyhogy az összes paraméter egymáshoz van igazítva. A Pythonban szereplő hurkok és lekérdezések beugrasztásától eltérően a Tkinter-objektumoknak ez a beugrasztása azonban nem kötelező.

```
Label(root, text="Bitte Button klicken, um die LED ein- und
auszuschalten").pack() (..."Kattintson a gombra a LED
bekapcsolása és kikapcsolása céljából")
```

A Label (címke) típusú objektumok tiszta szövegek egy grafikus elemben. Ezeket a program megváltoztathatja, azonban nem adnak lehetőséget párbeszédre a felhasználóval. Mindegyik Tkinter-objektum első paramétere a felülrendelt widget neve, ez többnyire azé az ablaké, amelyben a mindenkori objektum található. Esetünkben a programban csak egyetlen ablak van, a root-widget.

A text paraméter a címkén megjelenítendő szöveget tartalmazza. Az objektum definíciójának a végére az úgynevezett "packer" (becsomagoló) .pack() metódusként van hozzáfűzve. Ez a packer beépíti az objektumot a párbeszédablakba, és generálja a widget geometriáját.

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

A Button típusú objektumok kapcsolófelületek/nyomógombok, amelyekre a felhasználó rákattint egy bizonyos művelet kiváltása céljából. A textparaméter itt is azt a szöveget tartalmazza, amelynek meg kell jelennie a nyomógombon.

A command paraméter azt a funkciót tartalmazza, amelyet a nyomógombnak rákattintáskor le kell hívnia. Itt nem adhatók át paraméterek, és a funkcionévet zárójel nélkül kell megadni. Ez a nyomógomb a LedEin() funkciót hívja fel, amely bekapcsolja a LED-et.

A .pack() metódus tartalmazhat még paramétereket is, amelyek meghatározzák, hogyan helyezkedjen el egy objektum a párbeszédmezőn belül. A side=LEFT azt jelenti, hogy a nyomógomb balra igazítva, és ne középre igazítva legyen elhelyezve.

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

Ugyanezen séma szerint van elhelyezve még egy nyomógomb, amely a LedAus() funkción keresztül a LED-et
ismét kikapcsolja.

Most már definiálva van az összes funkció és objektum, így el lehet indítani a tulajdonképpeni programot.

root.mainloop() A főprogram csak egyetlenegy sorból áll. Ez indítja el a mainloop()főhurkot, a root-widget egy metódusát. Ez a programhurok arra vár, hogy a felhasználó működtessen egy widget-et, és ezzel beindítson egy műveletet.

A jobbra fent lévő, az ablak bezárására szolgáló x-szimbólumot nem kell a Tkinter által külön definiálni. Ha a felhasználó bezárja a root főablakot, automatikusan befejeződik a mainloop() főhurok.

GPIO.cleanup() A program tovább fut az utolsó sorig, és bezárja a megnyitott GPIO-portokat.

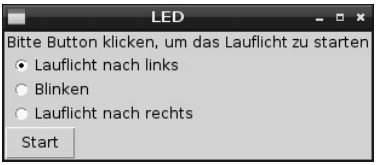
A program elindítása után megjelenik egy párbeszédmező a képernyőn. Kattintson rá az *Ein* nyomógombra a LED bekapcsolása, majd az *Aus* nyomógombra a kikapcsolása céljából.

10.2 Futófény vezérlése grafikus felülettel

A Tkinter Python-könyvtár még sokkal több vezérlőelemmel szolgál, mint egyszerű nyomógombokkal. A rádiógombokkal kiválasztó menüket hozhat létre, amelyekben a felhasználó a felajánlott több lehetőség közül egyet kiválaszthat.

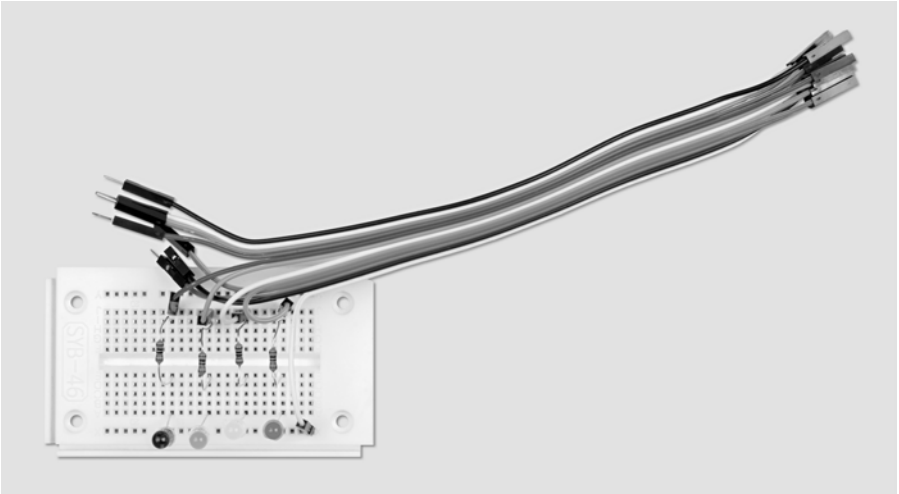
Mik a rádiógombok?
A »rádiógomb« név valójában a régi rádióktól származik, amelyeken az állomásgombok az adók előválasztásra szolgáltak. Ahányszor csak megnyomtunk közülük egyet, az utoljára megnyomott gombot egy rafinált mechanika kiugrasztotta. A rádiógombok ugyanígy viselkednek. Ha kiválasztunk egy opciót, a többi automatikusan kikapcsolódik.

A következő kísérlet különböző LED-villogási mintákat mutat be, amelyek hasonlítanak a "Tarka LED-minták és futófények" c. kísérlet mintáihoz. A különbség ahhoz képest annyi, hogy nem kell a felhasználónak számokat beírnia a szövegképernyőre, hanem kényelmesen egy egyszerű listából választhatja ki a kívánt fénymintát.



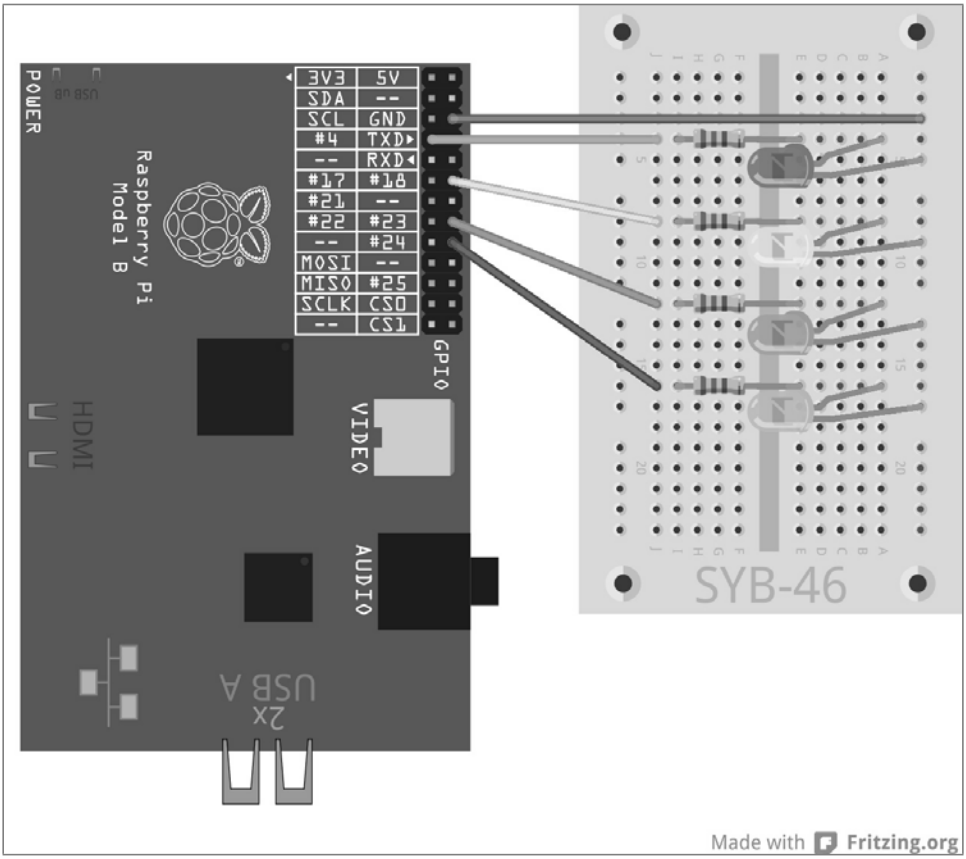
10.3 ábra 10.3: A párbeszédmező három LED-minta közötti választást kínál.

A kapcsolás felépítése azonos a "Tarka LED-minták és futófények" c. kísérlet felépítésével.



10.4 ábra 10.4: A dugasztábla beültetése a 10.2 kísérlethez.

A szükséges
alkatrészek: 1 db
dugasztábla
1 db piros
LED 1 db
sárga LED
1 db zöld
LED 1 db
kék LED
4 db 220-ohmos
ellenállás 5 db
összekötőkábel



10.5 ábra 10.5: Négy LED villog különféle fénymintákban.

A ledtk02.py program az előző programon alapszik, azonban ki lett bővítve a rádiógombokkal, továbbá a LED-futófények és a villogásminta számára készült funkcióval.

```
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 = [
    ("Lauflicht  nach  links",1),
    ("Blinken",2),(1. futófény
```

```
    ]
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="Bitte Button klicken, um das Lauflicht zu
starten").pack()
for txt, m in minta:
    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 Így működik

Az elején ismét a szükséges könyvtárak importálását végezzük. A legutóbbi programhoz képest a time-könyvtár is köztük van, amelyre a LED-villogási effektusok várakozási időihez van szükség.

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

Ezt követően a négy LED-hez készül egy lista. A megfelelő GPIO-portokat kimenetként definiáljuk, és 0-ra állítjuk, hogy az összes LED kezdéskor legyen kikapcsolva.

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

Két változó a program két értékét határozza meg: egy minta ismétléseinek a w számát, valamint egy minta t villogási idejét. Mindkét értéket a programban való előfordulásakor is fixen be lehetne írni. Ezen a módon azonban könnyebben lehet illeszteni őket, minthogy csak egy helyen vannak definiálva.

```
muster = [
    ("Lauflicht nach links",1), ("Blinken",2), ("Lauflicht nach rechts",3)
    ]
```

A választható három minta szövegét egy külön listaformátum definiálja. Mindhárom listaelem egy értékpárból áll, amely a kijelzett szöveget és egy számértéket tartalmaz, amelyet később az adott rádiógomb kiválasztásakor vissza kell adnia.

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

A root-widget inicalizálása megint megfelel az előző programénak, csak a párbeszédmező tartalmai különbözőek.

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

A Tk-párbeszédablakban alkalmazott változókat a normál Python változókkal ellentétben az első alkalmazás előtt deklarálni kell. Ez a két sor egy v változót egész számként deklarál, és a kezdéskor az 1értéket kapja.

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

Most ismét egy funkció definiálása következik, amelynek a neve a legutóbbi példával megegyezően LedEin() , ezúttal azonban nem csak egyetlen LED-et kapcsol be, hanem egy LED-mintát indít el. A legutóbbi példában szereplő LedAus()funkcióra itt nincs szükség. Az új funkció első sora kiolvassa a felhasználó által beadott v Tk-változót, és beírja az értéket az ePython-változóba. Hogy hogyan kerül az érték éppen a v változóba, megtudjuk alább a rádiógombok ismertetéséből.

A felhasználó választásától függően a három különböző programhurok közül az egyik elindul:

```
    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)
```

Az első esetben egy hurok ötször fut végig, amely a négy LED-et egymásután kigyújtja, 0,22 másodpercig hagyja világítani, majd megint kikapcsolja. Az öt ismétlést és a 0,2 másodperces felvillanási időt a program kezdetén a w és a t változó definiálja.

```
    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)
```

A második esetben mind a négy LED ötször egymásután egyszerre bekapcsolódik, és miután 0,2 másodpercig világított, megint egyszerre kikapcsolódik.

```
    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)
```

A harmadik eset megegyezik az elsővel, azzal a különbséggel, hogy a LED-ek számlálása visszafelé történik, és emiatt a futófény fordított irányban fut.

Miután a funkció definiálva lett, a grafikus felület elemei kerülnek elhelyezésre.

```
Label(root,
      text="Bitte Button klicken, um das Lauflicht zu
starten").pack() ("A futófény elindításához nyomja
```

A párbeszédmező szövege ismét Label-objektumként definiálódik. Új a három rádiógomb definiálása.

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

A rádiógombokat a for-hurok egy különleges formája definiálja. Egy hurokszámláló helyett két, párhuzamosan számlált változó van megadva. A két számláló egymásután átfut a musterlista elemein. Ekkor az első számlálóváltozó, a txt átveszi az értékekpár első értékét: a rádiógomb mellé kiírandó szöveget. A második számlálóváltozó, az m átveszi a mindenkori minta számát az egyes értékekpárok második értékéből.

A hurok ezen a módon elhelyezi a rádiógombokat, amelyeknek az első paramétere mindig a root , az a widget, amelyben a rádiógombok helyezkednek el. Egy rádiógomb textparamétere a kiírandó szöveget adja meg, amelyet ebben az esetben a txt változóból olvas ki. A variable paraméter egy előzőleg deklarált Tk-változót rögzít, amelybe a felhasználó választása után a kiválasztott rádiógomb értéke kerül bevitelre.

A value paraméter egy-egy számértéket határoz meg az egyes rádiógombok számára, amelyet ebben az esetben az m változóból olvas ki. Ha egy felhasználó rákattint erre a rádiógombra, a value paraméter értéke beíródik a variable alatt bevitt változóba. Mind a három rádiógomb definiálás után azonnal beépül a .pack() metódussal a párbeszédmezőbe. Az anchor=W paraméter arról gondoskodik, hogy a rádiógombok balra igazítva legyenek egymás alatt elhelyezve.

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

A Button nyomógomb a legutóbbi példával azonos módon van definiálva.

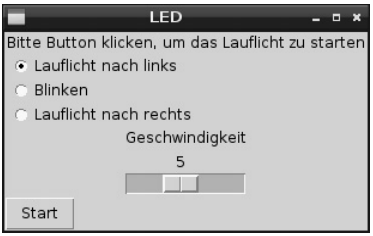
```
root.mainloop(); GPIO.cleanup()
```

A főhurok és a program befejezése is megegyezik a legutóbbi példával.

Indítsa el a programot, és válasszon ki az egyik rádiógombbal egy villogómintát. A v változón keresztül van az első választás előbeállítva. Ha a rádiógombokat egy párbeszédmezőben használja, mindig meg kell határozni egy értelmes előbeállítást, hogy ne kerülhessen sor definiálatlan eredményre, ha a felhasználó maga nem választ. A *Start* gombra való rákattintás után elindul a kiválasztott minta, és ötször lefut. Ezután választhat egy másik mintát.

10.3 A villogási sebesség beállítása

A harmadik lépésben még egyszer bővítjük a párbeszédmezőt. A felhasználó most egy tolószabályzóval beállíthatja a villogási sebességet.



10.6 ábra 10.6: Választható három LED-minta, és beállítható villogási sebesség.

A tolószabályzók alkalmazása
A tolószabályzók nagyon ösztönös módszert adnak egy adott tartományba eső számértékeknek a beadására. Ezzel a módszerrel megtakarítjuk a plauzibilitási lekérdezést, amely azt állapítaná meg, hogy a felhasználó beadott-e egy értéket, amelyet a program értelemszerűen át is tud alakítani, mivel a tolószabályzóval nem is lehet a tartományon kívül eső értékeket beadni. Állítsa mindig úgy a be a tolószabályzót, hogy az értékek a felhasználó számára értelmezhetők legyenek. Nincs semmi értelme a beállítást milliós nagyságrendben megengedni. Az abszolút számérték maga nem játszik tényleges szerepet, adjon a felhasználó kezébe egyszerűen egy 1-től 10-ig vagy 100-ig terjedő beosztású skálát, és számítsa megfelelően át az értéket a programban. Az értékek növekedjenek balról jobbra, fordítva a legtöbb felhasználó számára szokatlannak tűnne. Adjon be ezen kívül egy értelmes alapértéket, amelyet a program átvesz, ha a felhasználó nem változtatná meg a tolószabályzó állását.

A ledtk03.py program messzemenően megegyezik az előző példával, csupán a sebesség szabályozása jön hozzá.

```
import RPi.GIO 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 = [
    ("Lauflicht nach links",1), ("Blinken",2), ("Lauflicht nach rechts",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="Bitte Button klicken, um das Lauflicht zu
starten").pack()

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

Label(root,   text="Geschwindigket").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 Így működik

A könyvtárak és a GPIO-portok inicializálása, valamint a három villogóminta listájának a meghatározása megfelel az előző programénak. A villogási idő `t` változójának a meghatározása elmarad, mivel az később a tolószabályzóról kerül kiolvasásra.

`g = IntVar(); g.set(5)` Kiegészítésül a `v` Tk-változóhoz, amelyben a kiválasztott villogási minta tárolódik, még egy további egész számú változó, a `g` kerül deklarálásra a sebesség számára. Ennek a kezdeti értéke `5`, amely a tolószabályzó középértékének felel meg.

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

A LED-eket villogtató funkció ugyancsak megfelel az előbbi példának, azonban egy különbséggel. A villogás időtartamát meghatározó `t` változó a tolószabályzó `g` értékéből kerül meghatározásra.

Mivel a felhasználó a gyorsabb villogást ösztönösen a nagyobb sebességgel társítja, a tolószabályzó jobbra tolva nagyobb értékekeket szállít. A programban azonban a nagyobb sebességhez rövidebb várakozási időre, azaz kisebb értékre van szükség. Ezt reciprok érték képzéssel érjük el,

amely a tolószabályzó 1 -től 10 ig terjedő értékeire az 1.0 -től 0.1 -ig terjedő értékeket adja a t változóra. A képletben 1.0 -nek kell lennie és nem 1 -nek, hogy az eredmény lebegőpontos szám és ne egész szám legyen.

> Egész számok átszámítása lebegőpontos számmá
> Egy számítás eredménye automatikusan lebegőpontos számként kerül tárolásra, ha legalább egy érték a képletben lebegőpontos szám. Ha a képletben lévő minden érték egész szám (integer), akkor az eredmény ugyancsak egész számmá rövidül le.

A párbeszédmező címkéinek és rádiógombjainak a definíciója az előző példából lesz átvéve.

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

A tolószabályzó magyarázatára egy további címke kerül beírásra a párbeszédmezőbe. Mivel ennek nincs paramétere a pack() metódusban, vízszintesen középre igazítva kerül beépítésre a rádiógombok alá.

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

A tolószabályzó egy Scaletípusú objektum, amely az ebben a párbeszédmezőben lévő összes objektumhoz hasonlóan első paraméterként a root paramétert tartalmazza. Az orient=HORIZONTAL paraméter azt adja meg, hogy a tolószabályzó vízszintesen van elhelyezve. Enélkül a paraméter nélkül függőlegesen állna. A from_ és to (-tól -ig) paraméter a tolószabályzó kezdeti- és végértékét adja meg. Vegye figyelembe a from_ írásmódját, mivel a from alsó vonal nélkül a Pythonban rezervált szó a könyvtárak importálására. A variable paraméter meghatároz egy korábban deklarált Tk-változót, amelybe beírásra kerül a tolószabályzó pillanatnyi beállított értéke. A kezdeti érték a változó deklarációjakor meghatározott értékből van átvéve, amely ebben az esetben 5.

A tolószabályzó a pack()-metódussal most megint vízszintesen középre igazítva kerül beépítésre a párbeszédmezőbe.

A további programrészek – a *Start*-nyomógomb, a főhurok és a program befejezése – változatlanul van átvéve az előző példából.

Indítsa el a programot, válasszon ki egy villogómintát, és határozza meg a sebességet. Nagyobb értékek esetén a minta gyorsabban villog. Amikor rákattint a *Start*-nyomógombra, a LedEin() funkció kiolvassa a kiválasztott villogási mintát a rádiógombokról, valamint a sebességet a tolószabályzó állásából.

11 A PiDance LED-ekkel

A késői 70-es években, még az igazi számítógépes játékok előtt, volt egy elektronikus játék négy színes lámpával, amely 1979-ben még az év játéka rövidített listájára is rákerült. A játék Németországban *Senso* néven volt forgalomban. Az Atari licenciagyártásban forgalmazta *Touch Me* néven, zsebszámítógép-méretben. Egy további licenciagyártásra is sor került *Einstein* néven, míg angol nyelvterületen a Senso *Simon* néven került forgalomba.

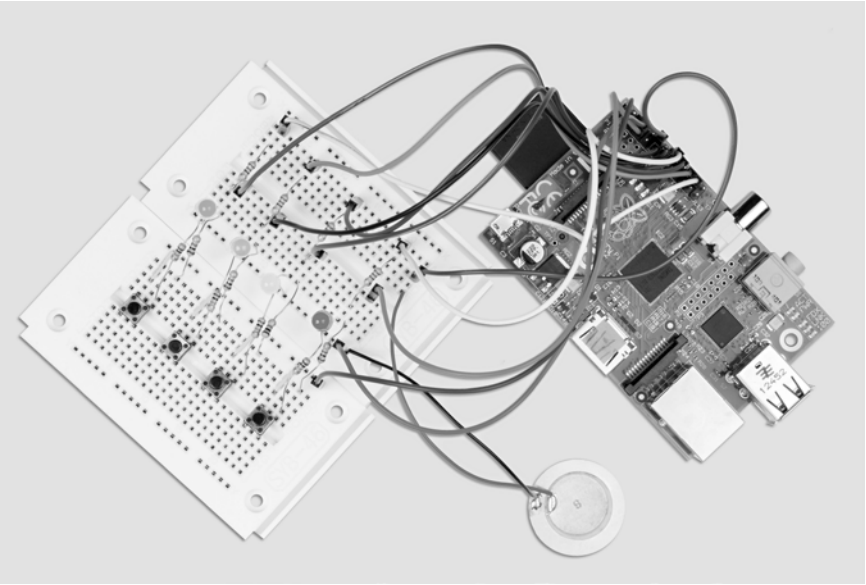
A Raspbian ennek a játéknak a grafikus verzióját forgalmazza a *Python Games* között *Simulate* néven.

11.1 ábra 11.1: A Simulate játék a Python Games-ből.

A PiDance nevű játékunk ugyancsak ezen a játékelven alapszik. LED-ek villognak véletlenszerű sorrendben. A felhasználónak ezután ugyanebben a sorrendben kell megnyomnia nyomógombokat. Minden egyes játékmenettel meggyullad egy további LED, úgyhogy egyre nehezebben lehet megjegyezni a sorrendet. Ha hibát követünk el, a játék befejeződik.

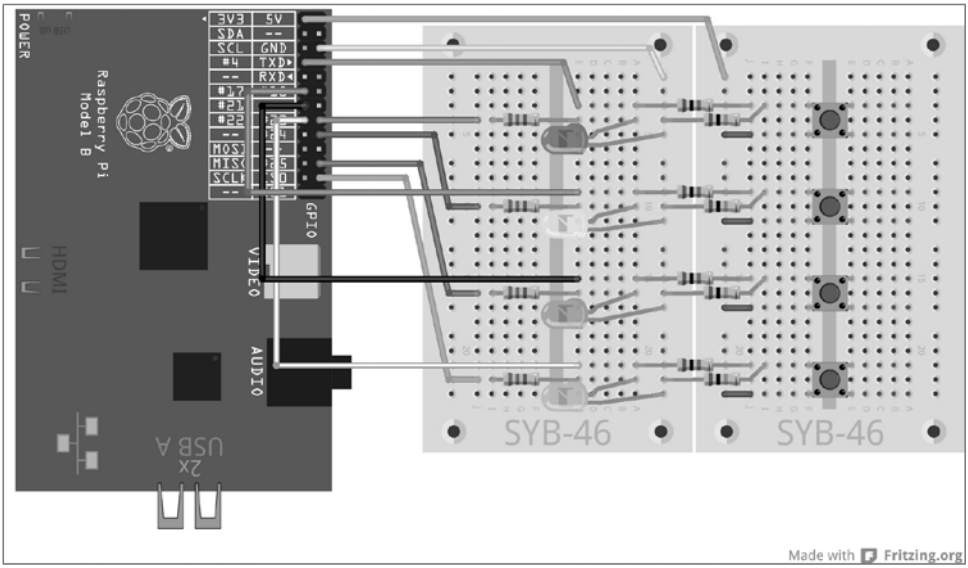
A játékot két dugasztáblán építjük fel úgy, hogy a nyomógombok a szélükön vannak, hogy jobban lehessen kezelni őket anélkül, hogy véletlenül kihúzzunk kábeleket a dugasztáblákból. A jobb stabilitás érdekében a dugasztáblák a hosszanti oldalukon összedughatók.

A már ismert összekötőkábeleken kívül még rövid huzaláthidalókra van szükség. Vágjon le ezért az együttszállított kapcsolóhuzalból egy csípőfogóval vagy drótvágó ollóval mintegy 2,5 centiméter hosszúságú darabokat, és távolítsa el mindkét végükről a szigetelést kb. 7 milliméter hosszúságban. Hajlítsa meg ezeket a huzaldarabokat U-alakra. Majd kössön össze velük két-két sort az egyik dugasztáblán.



11.2 ábra 11.2: A dugasztábla beültetése a 11. kísérlethez

A szükséges
alkatrészek: 2 db
dugasztábla
1 db piros
LED 1 db
sárga LED
1 db zöld
LED 1 db
kék LED
4 db 220-ohmos
ellenállás 4 db 1 kohmos
ellenállás 4 db 10
kohmos ellenállás 4 db
nyomógomb
10 db összekötőkábel
4 db rövid
huzaláthidaló



11.3 ábra 11.3: A PiDance LED-ekkel és nyomógombokkal két dugasztáblán.

A nyomógombok a hozzájuk tartozó LED-ekkel szemben vannak beépítve. A dugasztáblák összeköttetési helyénél lévő két-két középre eső hosszanti sor a kapcsolás számára a 0 V-os és a +3,3 V-os tápvezetékül szolgál.

A pidance01.py program a kész játékot tartalmazza.

```
# -*- 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 "Runde", 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 "Verloren!"
            print "Du hast es bis Runde", runde – 1, "geschafft"
            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 "Super gemacht!"
    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 Így működik

A programban sok új van, de a GPIO-vezérlés alapjai már ismertek.

`rzahl = 10` A `time`, a `random` és a `RPi.GPIO` modul importálása után egy `rzahl` változó kerül elhelyezésre, amely a játszandó fordulók számát határozza meg. Természetesen tíznél több fordulót is játszhat – minél több forduló, annál nehezebb megjegyezni a felvillanások sorrendjét.

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

A `farbe` lista egy hurkon keresztül annyi 0 és 3 közé eső véletlenszámmal töltődik fel, ahány fordulót játszunk. Ehhez az `append()` metódust alkalmazzuk, amely minden listában rendelkezésre áll. Ez a paraméterként átadott elemet hozzáfűzi a listához.

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

A LED-eket vezérlő GPIO-portok az ismert séma szerint egy LED listába vannak kimenetként elrendezve, és mind ki van kapcsolva.

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

Ugyanezen elv alapján vannak a négy nyomógomb GPIO-portjai egy TAST listába bemenetként elrendezve.

Ezzel az alapok el vannak intézve, és még két funkciót definiálunk, amelyre a programnak többször szüksége van.

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

A LEDein() funkció bekapcsol egy LED-et, és hagyja egy adott ideig világítani. Ez a funkció két paramétert alkalmaz. Az első paraméter, az n, adja meg a LED számát 0 és 3 között, a második paraméter, a z, adja meg azt az időt, ameddig a LED-nek világítania kell. Miután a LED újra kikapcsolódott, a funkció még 0,15 másodpercig vár, amíg kialszik, hogy többszöri felhívás esetén a LED-ek kigyulladása között látható legyen egy kis szünet. Ez különösen akkor fontos, ha egy LED többször egymásután kigyullad. Másképp ezt nem lehetne felismerni.

```
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
```

A Druecken() (megnyomni) funkció egy végtelen hurokból áll, amely arra vár, hogy a felhasználó megnyomjon egy gombot. Ezután a nyomógomb száma visszaadódik a főprogramba.

ok = True A funkció definálása után elindul a tulajdonképpeni főprogram, és elsőként egy ok változót a True állapotra állítja. Mihelyt a játékos egy hibát követ el, az ok változó a False állapotra állítódik. Ha a változó a megadott fordulószám után is még True, a játékos nyert.

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

A játék az rzahl változóban rögzített fordulón át folyik. A fordulószámláló 1-el el van tolva felfelé, mivel a játék az első (1) fordulóval kezdődik, és nem a nulladikkal (0).

```
print "Runde", runde
```

Az aktuális forduló a Python-Shell-ablakban jelenik meg.

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

Most a program lejátssza azt a mintát, amelyet a játékosnak meg kell jegyeznie. Az aktuális fordulószámnak megfelelően egymás után kigyullad sok LED a program kezdetén meghatározott farbe lista szerinti véletlenül kiválasztott színnel. Mivel a runde számláló 1-gyel kezdődik, már az első fordulóban világít pontosan egy LED. Ahhoz, hogy a LED világíthasson, a LEDein() funkciót alkalmazzuk, amelynek az első paramétere, a szín a megfelelő listaállásból való, a második paraméter mindegyik LED-et egy másodpercig hagyja világítani.

for i in range(runde): Miután a színminta lejátszódott, egy további hurok indul el, amelyben a játékosnak ugyanazt a mintát emlékezetből újra be kell adnia a nyomógombokkal.

taste = Druecken() Ehhez felhívódik a Druecken() funkció, amely addig vár, amíg a játékos meg nem nyom egy nyomógombot. A megnyomott nyomógombok száma a taste változóban tárolódik.

LEDein(taste, 0.2) Egy nyomógomb megnyomása után a megfelelő LED 0,2 másodpercre röviden felvillan.

if(taste != farbe[i]): Ha az utoljára megnyomott nyomógomb színe nem egyezik meg a lista megfelelő helyével, a játékos veszít. A != operátor a nem egyenlőt jelenti. Itt a <> is alkalmazható.

```
print "Verloren!" ("Veszített")
```

A program kiírja a képernyőre, hogy a játékos veszített, és hogy hány fordulóig jutott. A teljesített fordulók száma eggyel kisebb, mint a fordulószámláló pillanatnyi állása.

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

Optikailag látható jelzésül az összes LED bekapcsolódik ...

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

... majd egymás után 0,5 másodperces időközökben kikapcsolódik. Így egy látható leépítési hatás jön létre.

ok = False Az okváltozó, amely azt jelzi, hogy a játékos még játékban van-e, a False állapotra állítódik ...

break ... és a hurok megszakad. A játékos már nem nyomhat meg további nyomógombokat. Az első hibánál azonnal vége.

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

Ha az ok változó a False értéken áll, a külső hurok is megszakad, nincs további forduló már.

time.sleep(0.5) Ha a beadási sorrend helyes volt, a program 0,5 másodpercig vár, mielőtt elindítaná a következő fordulót.

if(ok == True): Ide akkor érkezik a program, ha vagy a hurok teljesen lefutott, a játékos tehát az összes szekvenciát helyesen adta be, vagy az előző hurok egy játékhiba miatt megszakadt. Ha az ok még a True állapoton állna, következik a győztes ünneplése. Máskülönben a program átugorja ezt a blokkot, és a játék még levezeti az utolsó programsort.

```
    print "Super gemacht!"
    for i in range(5):
    ("Szuper,
    megcsináltad!")
        for j in range(4):
            GPIO.output(LED[j], True)
        time.sleep(0.05)
```

Győzelem esetén megjelenik egy üzenet a Python-Shell-ablakban. Majd utána az összes LED egymásután ötször felvillan.

GPIO.cleanup() Az utolsó sor minden esetben kivitelre kerül. Itt bezáródnak az alkalmazott GPIO-portok.

Impresszum

© 2012 Franzis Verlag GmbH, Richard-Reitzner-Alle 2, 85540 Haar bei München

Szerző: Christian

Immler ISBN 978-3-

645-10145-5

Az elektromos és elektronikus készülékeket tilos a háztartási hulladékkal együtt eltávolítani. Az elhasznált terméket az érvényes törvényi előírásoknak megfelelően kell eltávolítani. Az elektromos készülékeit az eltávolítás céljára rendszeresített gyűjtőállomásokon
ingyenesen le lehet adni. Lakhelyén a helyi hatóságoknál informálódhat, hol talál ilyen gyűjtőállomást.

CE

A termék megfelel a vonatkozó CE irányelveknek, amennyiben azt a mellékelt használati útmutató szerint kezeli. A használati útmutató a termékhez tartozik, és vele kell adnia, ha a terméket továbbadja.

VIGYÁZAT! A szem védelme és a LED-ek:

Ne nézzen bele kis távolságból közvetlenül a LED fényébe, mert recehártya-gyulladást kaphat. Ez különösen az átlátszó házban lévő erős fényű LED-ekre vonatkozik, elsősorban az un. teljesítmény-LED-ekre. A fehér, a kék, az ibolya és az ultraibolya színű LED-ek látszólagos fényerőssége hamis benyomást ad a szem tényleges veszélyeztetettségéről. Főleg gyűjtőlencse használata esetén kell nagyon óvatosnak lenni. A LED-eket az útmutatóban megadott módon használja, de nagyobb áramfelvétellel ne.