

CE

**CONRAD**



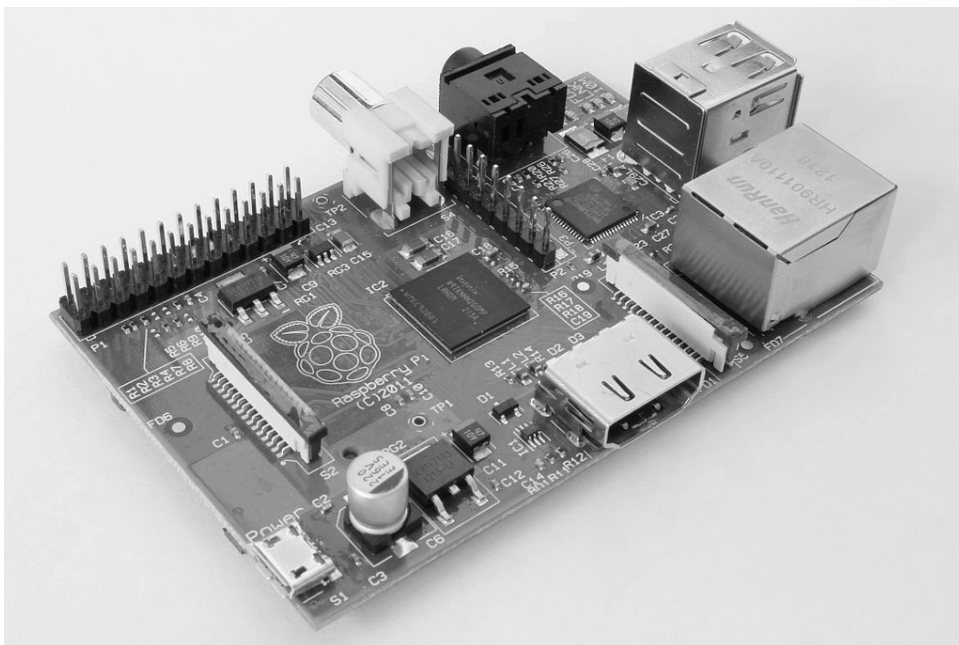
# Inhoudsopgave

<b>1</b>	<b>Van de installatie van het besturingssysteem tot en met het eerste Python-programma</b>	<b>5</b>
1.1	Wat heeft men nodig?	5
1.1.1	Micro-USB-oplader voor mobiele telefoon	6
1.1.2	Geheugenkaart	6
1.1.3	Toetsenbord	6
1.1.4	Muis	6
1.1.5	Netwerkkabel	6
1.1.6	HDMI-kabel	6
1.1.7	Audiokabel	7
1.1.8	Gele FBAS-videokabel	7
1.2	Raspbian-besturingssysteem installeren	7
1.2.1	Geheugenkaart in de PC voorbereiden	7
1.2.2	De software-installer NOOBS	8
1.2.3	De LED's op de Raspberry Pi.	8
1.2.4	De eerste start op de Raspberry Pi.	9
1.3	Bijna als Windows - de grafische interface LXDE	9
1.3.1	Eigen gegevens opslaan op de Raspberry Pi.	11
1.4	Het eerste programma met Python	12
1.4.1	Getallen raden met Python	14
1.4.2	Zo werkt het	16
<b>2</b>	<b>De eerste LED brandt op de Raspberry Pi</b>	<b>18</b>
2.1	Onderdelen in het pakket	19
2.1.1	Insteekprintplaten	20
2.1.2	Verbindingskabels	20
2.1.3	Weerstand en hun kleurcodes	21
2.2	LED aansluiten	22
2.3	GPIO met Python	26
2.4	LED aan- en uitschakelen	27
2.4.1	Zo werkt het	28
2.5	Python met GPIO-ondersteuning starten, zonder terminal	30
<b>3</b>	<b>Verkeerslicht</b>	<b>32</b>
3.1.1	Zo werkt het	34
<b>4</b>	<b>Voetangerslicht</b>	<b>36</b>
4.1.1	Zo werkt het	38
4.2	Toets op de GPIO-aansluiting	39
4.2.1	Zo werkt het	43
<b>5</b>	<b>Gekleurde LED-patronen en looplichten</b>	<b>45</b>
5.1.1	Zo werkt het	48

<b>6</b>	<b>LED via pulsduurmodulatie dimmen</b>	<b>53</b>
6.1.1	Zo werkt het	56
6.1.2	Twee LED's onafhankelijk van elkaar dimmen	57
6.1.3	Zo werkt het	59
<b>7</b>	<b>Ruimteweergave van geheugenkaart met LED's</b>	<b>60</b>
7.1.1	Zo werkt het	63
<b>8</b>	<b>Grafische dobbelsteen</b>	<b>65</b>
8.1.1	Zo werkt het	67
<b>9</b>	<b>Analoge klok op het beeldscherm</b>	<b>72</b>
9.1.1	Zo werkt het	73
<b>10</b>	<b>Grafische dialoogvelden voor de programmabesturing</b>	<b>77</b>
10.1.1	Zo werkt het	79
10.2	Looplicht met grafisch oppervlak besturen	81
10.2.1	Zo werkt het	84
10.3	Knippersnelheid instellen	87
10.3.1	Zo werkt het	88
<b>11</b>	<b>PiDance met LED's</b>	<b>89</b>
11.1.1	Zo werkt het	93

## 1 Van de installatie van het besturingssysteem tot en met het eerste Python-programma

Er is amper een elektronisch apparaat in zijn prijsklasse waar de afgelopen maanden zoveel over is gepraat als de Raspberry Pi. Ook wanneer u het op het eerste gezicht niet zou zeggen, is de Raspberry Pi een volwaardige computer ter grootte van ongeveer een kredietkaart - en dat voor een lage prijs. Zowel de hardware als de software zijn gunstig geprijsd: het besturingssysteem en alle dagelijks noodzakelijke toepassingen worden als gratis download aangeboden.



Afb. 1.1: De Raspberry Pi - een PC in een mini-formaat

Met de speciaal aangepaste Linux met grafisch oppervlak is de Raspberry Pi een stroombesparende, stille PC-variant. Zijn vrij programmeerbare GPIO-interface zorgt ervoor dat de Raspberry Pi uiterst interessant is voor hardware-bouwers en de maker-scene.

### 1.1 Wat heeft men nodig?

De Raspberry Pi is ondanks zijn zeer kleine formaat een volwaardige computer. Om hem te kunnen gebruiken, heeft men net als bij een »normale« PC nog toebehoren nodig - een besturingssysteem, voeding, netwerk, monitor, toetsenbord en verschillende aansluitkabels

### 1.1.1 Micro-USB-oplader voor mobiele telefoon

Voor de Raspberry Pi voldoet elke moderne netadapter voor mobiele telefoon. Oudere opladers uit de begintijd van de USB-oplaadtechniek zijn nog te zwak. Als men vermogensverbruikende USB-apparatuur aansluit zoals een harde schijven zonder eigen voeding, is een zwaardere netadapter noodzakelijk. De netadapter moet 5 V en minimaal 700 mA leveren, maar beter nog 1.000 mA. De ingebouwde vermogensregelaar voorkomt het »doorbranden« bij te zware netadapters.

#### **Op deze manier wordt een te zwakke netadapter kenbaar gemaakt**

Wanneer de Raspberry Pi wel boot, maar de muiscursor niet kan worden verplaatst of het systeem niet reageert op invoer van het toetsenbord, betekent dit dat er sprake is van een te zwakke voeding. Ook wanneer de toegang op aangesloten USB-sticks of harde schijven niet mogelijk is, moet een zwaardere netadapter worden gebruikt.

### 1.1.2 Geheugenkaart

De geheugenkaart dient in de Raspberry Pi als harde schijf. Zij bevat het besturingssysteem. Ook de eigen gegevens en geïnstalleerde programma's worden hierop opgeslagen. De geheugenkaart moet ten minste 4 GB geheugen hebben en volgens de informatie van de fabrikant minimaal class-4-norm ondersteunen. Deze norm geeft de snelheid van de geheugenkaart aan. Een actuele class-10-geheugenkaart onderscheidt zich duidelijk in de performance.

### 1.1.3 Toetsenbord

Elk gangbaar toetsenbord met USB-aansluiting kan worden gebruikt. Draadloze toetsenborden werken vaak niet, omdat ze teveel stroom of speciale drivers nodig hebben. Wanneer u niet over een ander toetsenbord beschikt, hebt u een USB-hub met aparte voeding nodig voor het gebruik van een radiogestuurd toetsenbord.

### 1.1.4 Muis

Een muis met USB-aansluiting is niet nodig wanneer men een besturingssysteem met grafische gebruikersinterface gebruikt op de Raspberry Pi. Sommige toetsenborden hebben extra USB-aansluitingen voor muizen, zodat ze geen andere aansluiting hoeven te bezetten. Deze kunnen dan bijv. voor een USB-stick worden gebruikt.

### 1.1.5 Netwerkkabel

Voor de verbinding met de router in het lokale netwerk is een netwerkkabel nodig. Voor de eerste inrichting is het in ieder geval nodig, later kan gebruik gemaakt worden van een WLAN. Zonder internettoegang zijn veel functies van de Raspberry Pi niet doelmatig bruikbaar.

### 1.1.6 HDMI-kabel

De Raspberry Pi kan met een HDMI-kabel aan monitoren of televisies worden aangesloten. Voor de aansluiting op computermonitoren met DVI-aansluiting zijn er speciale HDMI-kabels of adapters.

### **1.1.7 Audiokabel**

Via een audiokabel met 3,5 mm jackpluggen kunnen koptelefoons of PC-luidsprekers worden gebruikt op de Raspberry Pi. Het audiosignaal is tevens via de HDMI-kabel beschikbaar. Bij HDMI-televisies of monitoren is een audiokabel niet nodig. Wanneer een PC-monitor via een HDMI-kabel met DVI-adapter wordt aangesloten, gaat meestal op deze plaats het audiosignaal verloren, zodat de analoge audiouitgang opnieuw nodig is.

### **1.1.8 Gele FBAS-vidéokabel**

Wanneer er geen HDMI-monitor beschikbaar is, kan de Raspberry Pi met een analoge FBAS-vidéokabel met de typische gele stekkers, op een klassieke televisie worden aangesloten, waarbij de beeldschermresolutie echter zeer laag is. Voor televisies zonder gele FBAS-ingang zijn er adapters van FBAS naar SCART. De grafische interface kan in analoge televisieresolutie alleen met beperkingen worden bediend.

## **1.2 Raspbian-besturingssysteem installeren**

De Raspberry Pi wordt zonder besturingssysteem geleverd. Anders dan bij PC's, die bijna allemaal Windows gebruiken, wordt voor de Raspberry Pi een speciaal aangepast Linux geadviseerd. Windows kan bovendien op de sobere hardware helemaal niet lopen.

Raspbian is de naam van Linux-distributie, die door de fabrikant van de Raspberry Pi wordt geadviseerd en ondersteund. Raspbian is gebaseerd op Debian-Linux, een van de bekendste Linux-distributies, die onder andere de basis vormt van de populaire Linux-varianten Ubuntu en Knoppix. Wat bij PC's de harde schijf voorstelt, is bij de Raspberry Pi een geheugenkaart. Hierop bevinden zich het besturingssysteem en de gegevens, vanaf deze geheugenkaart boot de Raspberry Pi ook.

### **1.2.1 Geheugenkaart in de PC voorbereiden**

Aangezien de Raspberry Pi zelf nog niet kan booten, bereiden we de geheugenkaart op de PC voor. Hiervoor heeft u een kaartlezer nodig aan de PC. Deze kan vast ingebouwd of via USB worden aangesloten.

Gebruik bij voorkeur de vanaf fabriek geleverde geheugenkaarten, omdat deze door de fabrikant al optimaal vooraf zijn geformatteerd. U kunt echter ook een geheugenkaart gebruiken, die al eerder in een digitale camera of een ander apparaat is gebruikt. Deze geheugenkaarten moeten voor gebruik voor de Raspberry Pi opnieuw worden geformatteerd. Theoretisch kunt u hiervoor de formatteringsfuncties van Windows gebruiken. De software »SDFormatter« van de SD Association is echter duidelijk beter. Hiermee worden de geheugenkaarten voor optimale performance geformatteerd. Deze tool kan onder [www.sdcard.org/downloads/formatter\\_4](http://www.sdcard.org/downloads/formatter_4) gratis worden gedownload.

Indien de geheugenkaart partities van eerdere besturingssysteeminstallaties bevat, wordt niet de volledige grootte aangegeven in de SDFormatter. Gebruik in dit geval de formatteringsoptie *FULL (Erase)* en schakel de optie *Format Size Adjustment* in. Hiermee wordt de partitionering van de geheugenkaart opnieuw aangemaakt

## Geheugenkaart wordt gewist

Gebruik bij voorkeur een lege geheugenkaart voor de installatie van het besturingssysteem. Als er zich op de geheugenkaart gegevens bevinden, worden deze door de nieuwe formattering tijdens de besturings-systeeminstallatie onherroepelijk gewist.

### 1.2.2 De software-installer NOOBS

»New Out Of Box Software« (NOOBS) is een uiterst eenvoudige installer voor Raspberry Pi-besturingssystemen. Hier hoeft de gebruiker zich niet meer, zoals eerder, bezig te houden met image-tools en bootblokken, om een boot-capabele geheugenkaart in te richten. NOOBS biedt verschillende besturingssystemen om uit te kiezen, waarbij men bij de eerste start direct het gewenste besturingssysteem kan kiezen op de Raspberry Pi, dat dan boot-capabel wordt geïnstalleerd op de geheugenkaart. Download het ongeveer 1,2 GB grote installatiearchief voor NOOBS op de officiële downloadpagina [www.raspberrypi.org/downloads](http://www.raspberrypi.org/downloads) en pak het op de PC uit op een minimaal 4 GB grote geheugenkaart.

Start nu de Raspberry Pi met deze geheugenkaart. Steek hem hiervoor in het slot van de Raspberry Pi en sluit het toetsenbord, muis, monitor en netwerkkabel aan. De USB-stroomaansluiting komt als laatste. Hiermee wordt de Raspberry Pi aangezet. Er is geen aparte aanzetknop.

Na enkele seconden verschijnt een keuzemenu, waarin het gewenste besturingssysteem kan worden gekozen. Wij gebruiken het door de Raspberry Pi-stichting geadviseerde besturingssysteem Raspbian.

Kies helemaal onderaan Nederlands als installatietaal en markeer het vooraf gekozen Raspbian-besturingssysteem. Na bevestiging van de veiligheidsmelding, dat de geheugenkaart wordt overschreven, start de installatie, die enkele minuten duurt. Tijdens de installatie wordt korte informatie over Raspbian weergegeven.

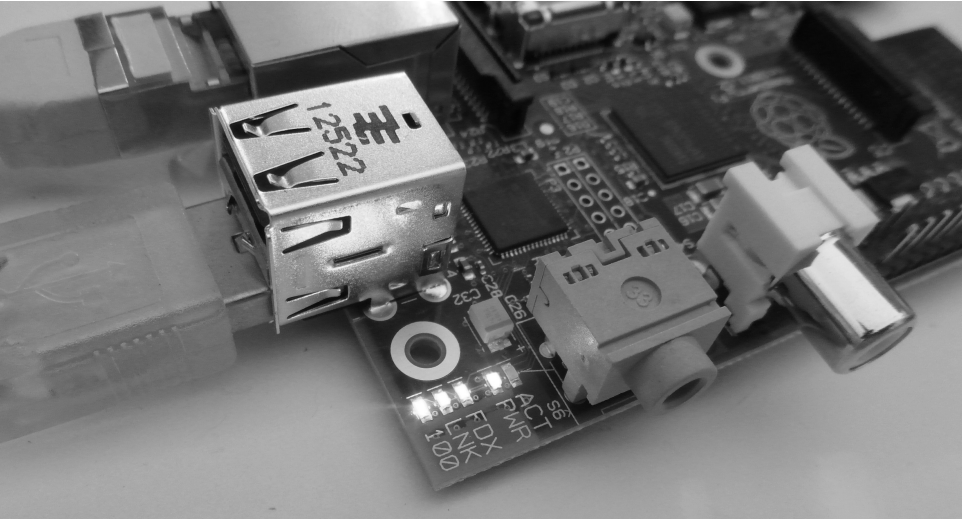
### 1.2.3 De LED's op de Raspberry Pi.

In een hoek op de Raspberry Pi bevinden zich vijf LED's met statusaanduidingen. De aanduidingen zijn op nieuwere en oudere Raspberry Pi-modellen gedeeltelijk verschillend, die functies zijn echter gelijk.

Nieuwe printplaat (rev. 2)	Printplaat (rev. 1 ouder)	Kleur	Betekenis van de LED
ACT	OK	Groen	Toegang tot de geheugenkaart
PWR	PWR	Rood	Verbonden met stroomtoevoer
FDX	FDX	Groen	LAN in de volledige duplexmodus
LNK	LNK	Groen	Toegang tot de LAN
100	10M	Geel	LAN met 100 Mb/s

Tab. 1.1: De LED's op de Raspberry Pi.





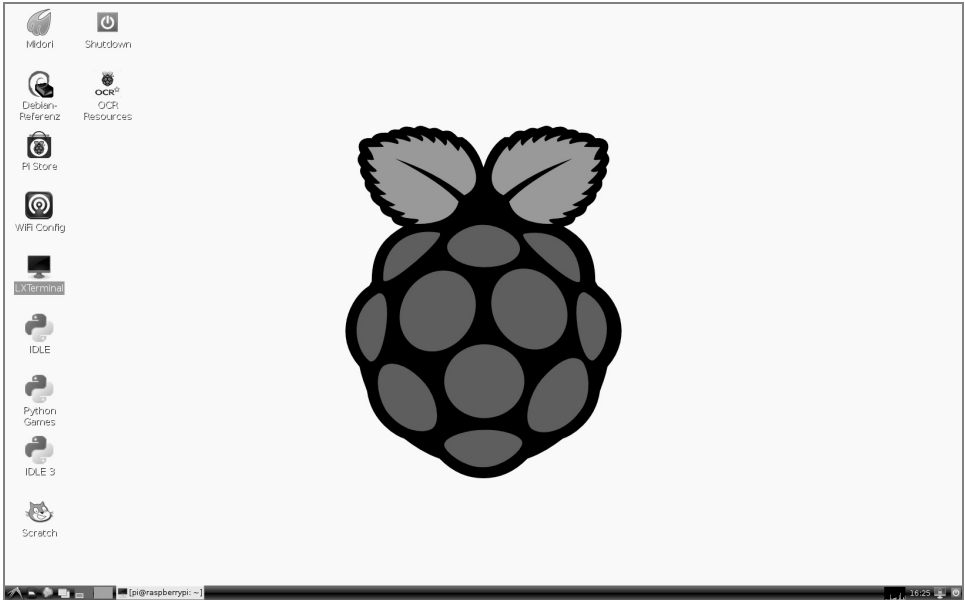
Afb. 1.2: De status-LED's op de Raspberry Pi.

#### 1.2.4 De eerste start op de Raspberry Pi.

Na afsluiting van de installatie boot de Raspberry Pi opnieuw en start automatisch de configuratie-tool `raspi-config`. Hier hoeft u enkel nog onder *Enable Boot to Desktop* de optie *Desktop Log in as user 'pi'* te kiezen. De Nederlandse taal en Nederlandse toetsenbordindeling zijn overeenkomstig overige belangrijke basisinstellingen al automatisch gekozen. Na opnieuw opstarten is de grafische LXDE-desktop beschikbaar.

#### 1.3 Bijna als Windows - de grafische interface LXDE

Velen schrikken bij het woord eerst even terug, omdat ze bang zijn, cryptische opdrachtvolgordes via commandoregels te moeten invoeren, net als 30 jaar geleden onder DOS. Integendeel! Linux biedt als open besturingssysteem voor ontwikkelaars vrije mogelijkheden, eigen grafische interfaces te ontwikkelen. Men zit als gebruiker van de in hoofdzaak nog commandoregelsgeoriënteerde besturingssysteem niet vast aan één interface.



**Afb. 1.3:** De LXDE-desktop op de Raspberry Pi komt in grote mate overeen met Windows XP.

Raspbian-Linux voor de Raspberry Pi gebruikt de interface LXDE (Lightweight X11 Desktop Environment), die enerzijds zeer weinig systeemressourcen nodig heeft en anderzijds met zijn startmenu en de verkenner in grote mate overeenkomt met de bekende Windows-interface.

### Linux-aanmelding

Ook de bij Linux kenmerkende gebruikersaanmelding wordt op de achtergrond uitgevoerd. Mocht u het toch een keer nodig hebben: De gebruikersnaam is `pi` en het wachtwoord `raspberrypi`.

Het LXDE-pictogram helemaal linksonder opent het startmenu, de pictogrammen ernaast zijn de verkenner en de webbrowser. Het startmenu is net als onder Windows meerfasig opgebouwd. Regelmatig gebruikte programma's kunnen met een rechter muisklik op de desktop worden gezet. Hier bevinden zich al enkele van de vooraf geïnstalleerde programma's, de Midori-webbrowser, Python-ontwikkelingsplatformen en de Pi store.

### Raspberry Pi uitschakelen

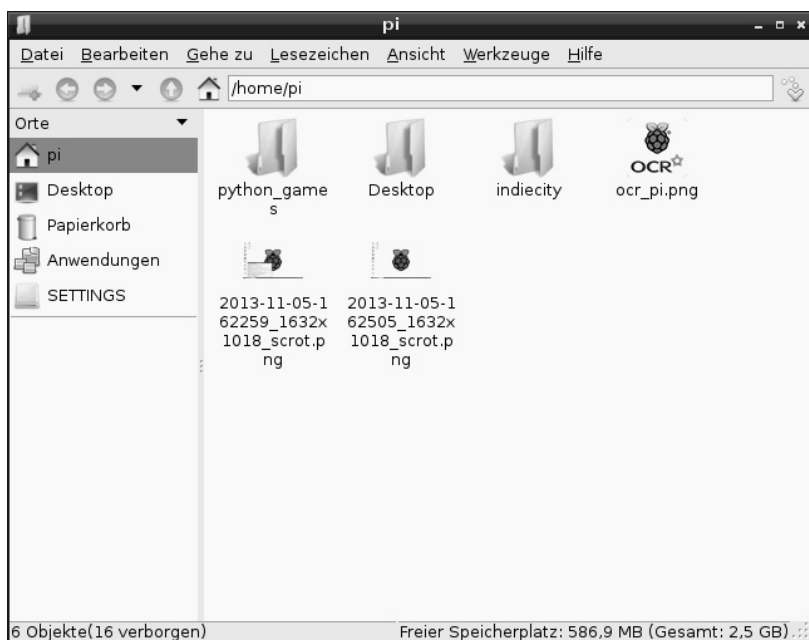
Theoretisch gezien kan men bij de Raspberry Pi eenvoudig de stekker eruit trekken, en dan schakelt hij uit. Het is echter beter, het systeem net als op een PC netjes af te sluiten. Dubbelklik hiervoor op de desktop op het *Shutdown*-pictogram.

### 1.3.1 Eigen gegevens opslaan op de Raspberry Pi.

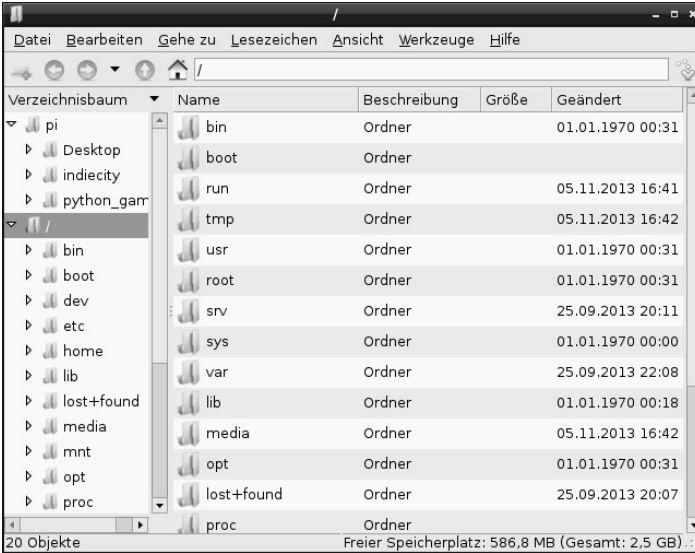
Het bestandsbeheer loopt onder Linux echter anders dan onder Windows, maar is echter niet moeilijker. Raspbian komt met een verkenner, die sprekend op Windows-Explorer lijkt. Een belangrijk onderscheid ten opzichte van Windows: Linux scheidt niet strikt volgens loopwerken, alle bestanden bevinden zich in een gezamenlijk bestandssysteem.

Onder Linux plaatst men alle eigen bestanden principieel alleen onder de eigen home-map. Hier wordt het `/home/pi` genoemd, volgens de gebruikersnaam `pi`. Linux gebruikt de eenvoudige schuine streep voor de scheiding van mapniveaus(/), niet de van Windows bekende Backslash (\). In deze map gaat u ook uw Python-programma's onderbrengen. De verkenner, die men behalve via het startmenu tevens net als onder Windows met de toetsencombinatie `Win + E` kan starten, geeft standaard ook alleen deze home-map weer. Sommige programma's maken daar automatisch submappen aan.

Wie echt alles wil zien, ook de bestanden, waarmee de gebruiker niets van doen heeft, schakelt de verkenner linksboven om van *Orte* naar *mappenboom*. Kies dan in het menu nog onder *Weergave* de optie *Detailweergave*, en de weergave ziet er uit zoals met zich dat bij Linux voorstelt.



**Afb. 1.4:** De verkenner op de Raspberry Pi kan er op deze manier uitzien ...



Afb. 1.5: ... of op deze manier.

### Hoeveel vrije ruimte heeft de geheugenkaart?

Niet alleen harde schijven van PC's zijn voortdurend vol - bij de geheugenkaart van de Raspberry Pi kan dit nog veel sneller gebeuren. Des te belangrijker is het, de vrije en bezette ruimte op de geheugenkaart altijd in de gaten te houden. De statusregel van de verkenner onderin de beeldschermrand, geeft rechts de vrije en bezette opslagruimte weer op de geheugenkaart.

## 1.4 Het eerste programma met Python

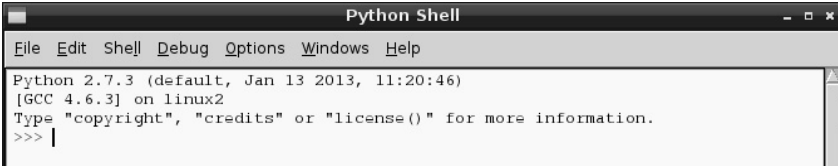
Om met de programmering te beginnen is op de Raspberry Pi de programmeertaal Python vooraf geïnstalleerd. Python overtuigt door zijn heldere structuur, die een eenvoudig begin in het programmeren toestaat, maar is tevens een ideale taal, om iets »even snel« te automatiseren, wat men anders handmatig zou doen. Omdat er geen variabelendeclaraties, types, klassen of moeilijke regels hoeven te worden opgevolgd, is het programmeren echt leuk.

### Python 2.7.3 of 3.3.0?

Op de Raspberry Pi zijn direct twee versies van Python vooraf geïnstalleerd. Helaas gebruikt de nieuwste Python-versie 3x gedeeltelijk een andere syntax als de vertrouwde versie 2 x, zodat programma's van de ene versie niet met de andere lopen. Enkele belangrijke bibliotheken, zoals bijv. de bekende PyGame voor het programmeren van spellen en grafische beeldschermuitvoeren in het algemeen, zijn nog niet beschikbaar voor Python 3 x. Daarom, en omdat ook de meeste in het internet beschikbare programma's voor Python 2 x zijn geschreven, gebruiken we in dit boek de vertrouwde Python-versie 2.7.3. Indien op uw Raspberry Pi een oudere Python-versie met een versienummer 2 x is geïnstalleerd, werken onze voorbeelden hiermee op dezelfde manier.



Python 2.7.3 wordt met het pictogram *IDLE* op de desktop gestart. Hier verschijnt een op het eerste gezicht eenvoudig invoerscherm met een opdrachtprompt.



Afb. 1.6: Het invoerscherm van de Python-shell.

In dit scherm opent u aanwezige Python-programma's, schrijft u nieuwe of kunt u ook direct Python-commando's interactief verwerken, zonder een eigenlijk programma te moeten schrijven. Voer bijv. bij de prompt het onderstaande in:

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

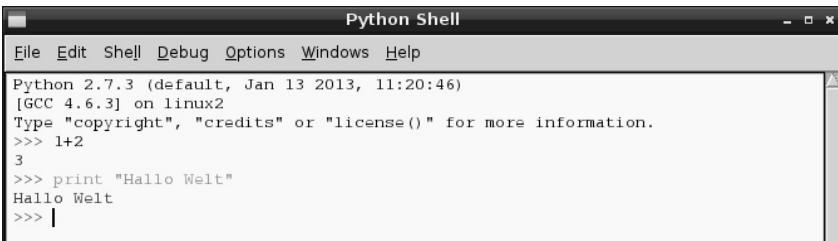
Dan verschijnt onmiddellijk het juiste antwoord.

```
3
```

Op deze manier kan Python als handige zakrekenmachine worden gebruikt, maar dit heeft nog niet met programmering te maken. Gewoonlijk beginnen programmeercursussen met een *Hallo wereld*-programma, dat »Hallo wereld« op het beeldscherm schrijft. Dit is in Python dermate eenvoudig, dat het niet eens loont, hiervoor een eigen opschrift in te voegen. Typ in het Python-shell-scherm eenvoudig de volgende regel:

```
>>> print "Hallo wereld"
```

Dit eerste »programma« schrijft dan *Hallo wereld* in de volgende regel op het beeldscherm.



Afb. 1.7: »Hallo wereld« in Python (bovenaan is nog de uitvoer van de berekening zichtbaar).

Hier ziet u tevens direct, dat de Python-shell ter verduidelijking automatisch verschillende tekstkleuren gebruikt. Python-commando's zijn oranje, tekenreeksen groen en resultaten blauw. Later gaat u nog meer kleuren ontdekken.

## Python-flashcards

Python is de ideale programmeertaal, om de instap in de programmering te leren. Men moet alleen wennen aan de syntax en de layout-regels. Ter assistentie in de dagelijkse programmering worden de belangrijkste syntaxelementen van de Python-taal in de vorm van »spiekbriefjes« kort beschreven. Deze zijn gebaseerd op de Python-flashcards van David Whale. Wat dit precies betekent, vindt u onder [bit.ly/pythonflashcards](http://bit.ly/pythonflashcards). Deze flashcards geven geen uitleg over de technische achtergronden, maar beschrijven alleen aan de hand van zeer korte voorbeelden de syntax, dus hoe iets wordt gedaan.

### 1.4.1 Getallen raden met Python

In plaats van ons met programmatheorie, algoritmes en gegevenstypes bezig te houden, schrijven we direct het eerste kleine spel in Python, een eenvoudig raadspel, waarin een door de computer willekeurig gekozen getal in zo min mogelijk stappen moet worden geraden.

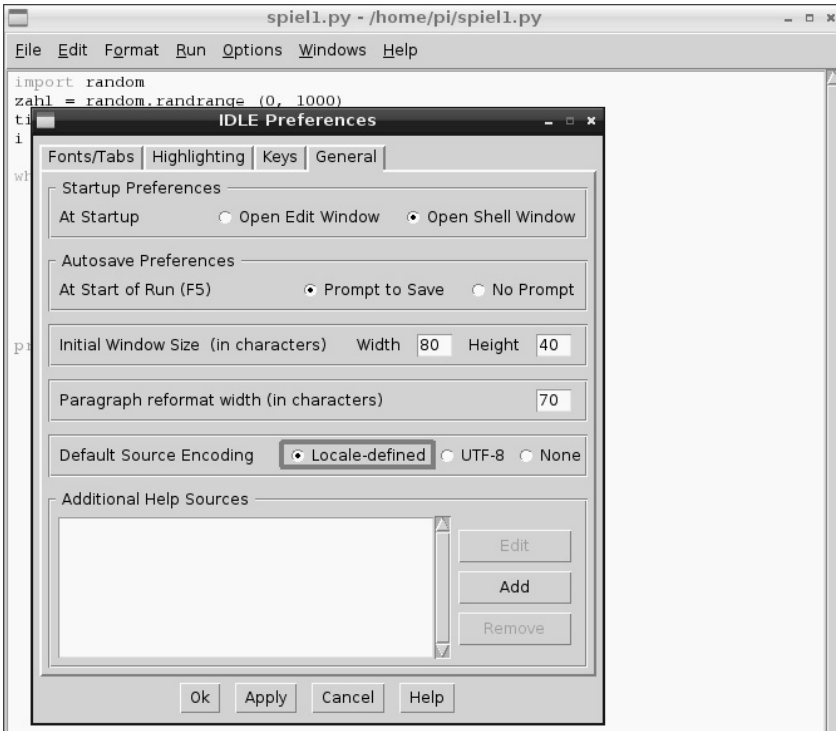
1. Kies in het menu van de Python-shell *File/New Window*. Hier wordt een nieuw scherm geopend, waarin de onderstaande programmacode moet worden getypt:

```
import random
getal = random.randrange(1000); tipp = 0; i = 0
while tipp != getal:
    tipp = input("Jouw tip:")
    if getal < tipp:
        print "Het gezochte getal is kleiner dan ",tipp

    if getal > tipp:
        print "Het gezochte getal is groter dan ",tipp

    i += 1
print "Je hebt het getal bij de ",i,". tip geraden"
```

2. Sla het bestand op via *File/Save As* als `spell.py`. Of u download het kant-en-klare programmabestand onder [www.buch.cd](http://www.buch.cd) en opent het in de Python-shell met *File/Open*. De kleurcodering in de brontekst verschijnt automatisch en helpt, om typefouten te vinden.
3. Voordat u het spel start moet u nog rekening houden met een specifiek onderdeel van de Nederlandse taal, namelijk de dubbele punt. Python loopt op verschillende computerplatformen, die dubbele punten verschillend coderen. Om er voor te zorgen dat ze juist worden weergegeven, kiest u in het menu *Options/Configure IDLE* en schakel op de registerkaart *General* de optie *Locale-defined* in het bereik *Default Source Encoding* in.



Afb. 1.8: De juiste instelling voor de weergave van dubbele punten in Python.

4. Start nu het spel met de toets `[F5]` , of het menupunt *Run/Run Module*.
5. Het spel ziet omwille van de eenvoud af van elke grafische interface, alsmede op verklarende teksten of plausibiliteitsvragen voor de invoer. De computer genereert op de achtergrond een willekeurig getal tussen 0 en 1.000. Vermeld eenvoudig een tip en u verneemt, of het gezochte getal groter is of kleiner. Door meerdere tips komt u bij het juiste getal uit.

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Dein Tipp:300
Die gesuchte Zahl ist größer als 300
Dein Tipp:800
Die gesuchte Zahl ist kleiner als 800
Dein Tipp:500
Die gesuchte Zahl ist kleiner als 500
Dein Tipp:450
Die gesuchte Zahl ist kleiner als 450
Dein Tipp:350
Die gesuchte Zahl ist kleiner als 350
Dein Tipp:320
Die gesuchte Zahl ist größer als 320
Dein Tipp:330
Die gesuchte Zahl ist kleiner als 330
Dein Tipp:325
Die gesuchte Zahl ist kleiner als 325
Dein Tipp:322
Du hast die Zahl beim 9 . Tipp erraten
>>> |
```

Afb. 1.9: Getallen raden in Python

### 1.4.2 Zo werkt het

Dat het spel werkt, kan eenvoudig worden uitgeprobeerd. Nu ontstaan er natuurlijk enkele vragen: Wat gebeurt er op de achtergrond? Wat betekenen de losse programmeregels?

`import random` Om het willekeurige getal te genereren, wordt een externe Python-module met de naam `random` geïmporteerd, dat verschillende functies voor willekeursgeneratoren bevat.

`getal = random.randrange(1000)` De functie `randrange` uit de module `random` genereert een willekeurig getal in het door de parameter begrensde getallenbereik, hier tussen 0 en 999. De parameter van de functie `random.randrange()` geeft het aantal mogelijke willekeurige getallen, beginnend met 0, dus altijd het eerste getal dat niet wordt bereikt. Hetzelfde geldt ook voor lussen en overeenkomstige functies in Python.

Dit willekeurige getal wordt in de variabelen `getal` opgeslagen. Variabelen zijn in Python geheugenplaatsen, die een willekeurige naam hebben en getallen, tekenreeksen, lijsten of andere gegevenstypen kunnen opslaan. Anders dan in sommige andere programmeertalen moet ze niet vooraf worden gedeclareerd.



### Hoe ontstaan willekeurige getallen?

Over het algemeen wordt gedacht dat in een programma niets toevallig kan gebeuren. Hoe kan het dan, dat een programma willekeurige getallen kan genereren? Wanneer een groot priemgetal door een willekeurige waarde wordt gedeeld, ontstaan vanaf de  $x$ -te plaats na de komma getallen, die nauwelijks nog voorspelbaar zijn. Deze veranderen ook zonder enige regelmatigheid, wanneer men de divisor regelmatig verhoogt. Dit resultaat is dan wel ogenschijnlijk toevallig, maar kan echter door een identiek programma of herhaaldelijk oproepen van hetzelfde programma op elk moment worden gereproduceerd. Neemt men echter een uit enkele van deze cijfers opgebouwd getal en deelt men dat weer door een getal, dat resulteert uit de actuele tijdseconde, of de inhoud van een willekeurige opslagplaats van de computer, verschijnt een resultaat, dat niet kan worden gereproduceerd en daarom als willekeurig getal wordt gekenmerkt.

`tip=0` De variabele `tip` bevat later het getal die de gebruiker als `tip` invoert. Aan het begin is ze 0.

`i=0` De variabele `i` heeft zich onder programmeurs als teller van programmalusdoorlopen ingeburgerd. Hier wordt ze gebruikt, om het aantal tips te tellen, die de gebruiker heeft gebruikt, om het geheime getal te raden. Ook deze variabele staat aan het begin op 0.

`while tip != getal:` Het woord `while` (Engels voor »terwijl«) leidt een programmalus in, die in dit geval zolang wordt herhaald, als `tip`, het getal, dat de gebruiker als `tip` invoert, en niet gelijk is aan het geheime getal `zahl`. Python gebruikt de tekencombinatie `!=` voor niet gelijk aan. Achter de dubbele punt volgt de eigenlijke programmalus.

`tip = input("Jouw tip:")` De functie `input` schrijft de tekst `Jouw tip:` en verwacht hiernaar een invoer, die in de variabelen `tip` wordt opgeslagen.

### Inspringing is belangrijk in Python

In de meeste programmeertalen worden programmalussen of beslissingen ingesprongen, om de programmacode overzichtelijker te maken. In Python dienen deze inspringingen niet alleen voor overzichtelijkheid, maar zijn tevens nodig voor de programmalogica. Hiervoor heeft men hier geen speciale leestekens nodig, om lussen of beslissingen te beëindigen.

`if getal < tip:` Wanneer het geheime getal `zahl` kleiner is dan de door de gebruiker getipte getal `tip`, dan ...

```
print "Het gezochte getal is kleiner dan ",tip
```

... verschijnt deze tekst. Aan het einde staat hier de variabele `tip`, opdat het getipte getal in de tekst wordt weergegeven. Is deze voorwaarde niet van toepassing, wordt de ingesprongen regel gewoon gepasseerd.

`if tip < getal:` Wanneer het geheime getal `zahl` groter is dan het door de gebruiker getipte getal `tip`, dan ...

```
print "Het gezochte getal is groter dan ",tip
```

.... verschijnt een andere tekst.

`i += 1` In elk geval - daarom niet meer ingesprongen - wordt de teller `i`, die de pogingen telt, met 1 verhoogd. Deze regel met de operator `+=` betekent hetzelfde als `i = i + 1`.

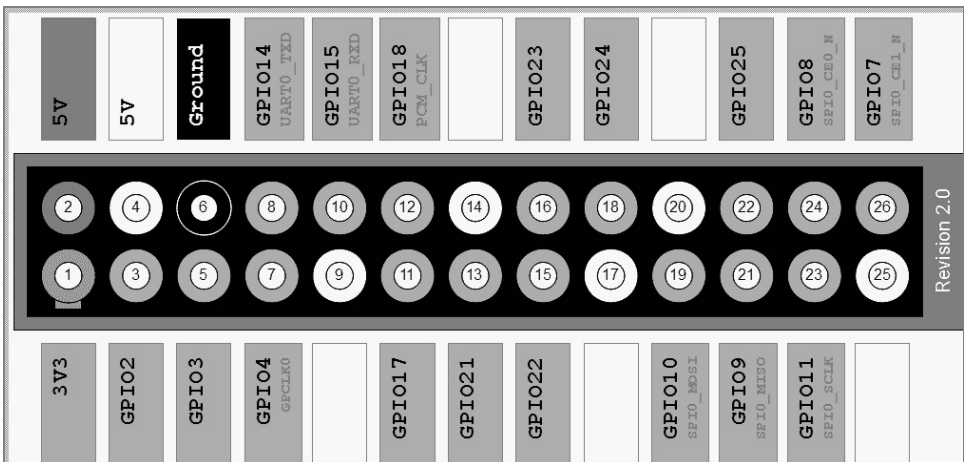
```
print "Je hebt het getal bij de ",i,". tip geraden"
```

Deze regel is meer ingesprongen en dit betekent dat ook de `while`-lus is beëindigd.. Als deze voorwaarde niet meer van toepassing is, is dus het door de gebruiker getipte getal `tip` niet meer niet gelijk aan (maar gelijk aan) het geheime getal `zahl`, wordt deze tekst weergegeven, die uit twee zinsdelen en de variabelen `i` bestaat en zo aangeeft, hoeveel pogingen de gebruiker nodig had.. Python-programma's hebben geen eigen aanwijzing nodig om te beëindigen. Ze eindigen eenvoudig na de laatste opdracht of na een `break`, die niet meer wordt uitgevoerd en die geen verdere opdrachten opvolgen.

## 2 De eerste LED brandt op de Raspberry Pi

De 26-polige contactstrip in de hoek van de Raspberry Pi biedt de mogelijkheid, om direct hardware aan te sluiten, om bijv. via toetsen in te voeren of programmeerbaar LED's te laten branden. Deze contactstrip wordt als GPIO gekenmerkt. De Engelse afkorting »General Purpose Input Output« betekent in het Nederlands gewoon »Algemene in- en uitvoer«.

Van deze 26 pinnen kunnen 17 naar keuze als in- of uitgang worden geprogrammeerd en zo voor een groot aantal hardware-uitbreidingen worden gebruikt. De overigen zijn vast ingesteld voor de voeding en andere doeleinden.



Afb. 2.1: Bezetting van de GPIO-interface De grijze lijn boven en links kenmerkt de rand van de printplaat GPIO-pin 2 bevindt zich dus in de buitenhoek van de Raspberry Pi.

### **Voorzichtig**

Verbind in geen geval enigerlei GPIO-pinnen met elkaar om daarna te wachten, wat er gebeurt, maar volg absoluut de onderstaande aanwijzingen op:

enkele GPIO-pinnen zijn direct met aansluitingen van de processor verbonden, kortsluiting kan de Raspberry Pi volledig kapot maken. Verbindt men twee pinnen met elkaar via een schakelaar of een LED, moet hiertussen altijd een voorweerstand worden geschakeld.

Gebruik voor logicasignalen altijd pin 1 die +3,3 V levert en tot 50 mA kan worden belast. Pin 6 is de aardkabel voor logicasignalen. De andere, met *Ground of 3V3* gekenmerkte pinnen 9, 14, 17, 20, 25 zijn bedoeld voor toekomstige uitbreidingen. Ze kunnen zoals nu beschreven worden gebruikt. Doe dit echter niet, wanneer u de eigen projecten ook op toekomstige Raspberry Pi-versies wilt kunnen gebruiken.

Elke GPIO-pin kan als uitgang (bijv. voor LED's) of als ingang (bijv. voor toetsen) worden geschakeld. GPIO-uitgangen leveren in de logicatoestand *1* een spanning van +3,3 V, in de logicatoestand *0* 0 Volt. GPIO-ingangen leveren bij een spanning tot +1,7 V het logicasignaal *0*, bij een spanning tussen +1,7 V en +3,3 V het logicasignaal *1*.

Pin 2 levert +5 V voor de voeding van externe hardware. Hier kan zoveel stroom worden onttrokken, als de USB-netadapter van de Raspberry Pi levert. Deze pin mag echter niet met een GPIO-ingang worden verbonden.

## **2.1 Onderdelen in het pakket**

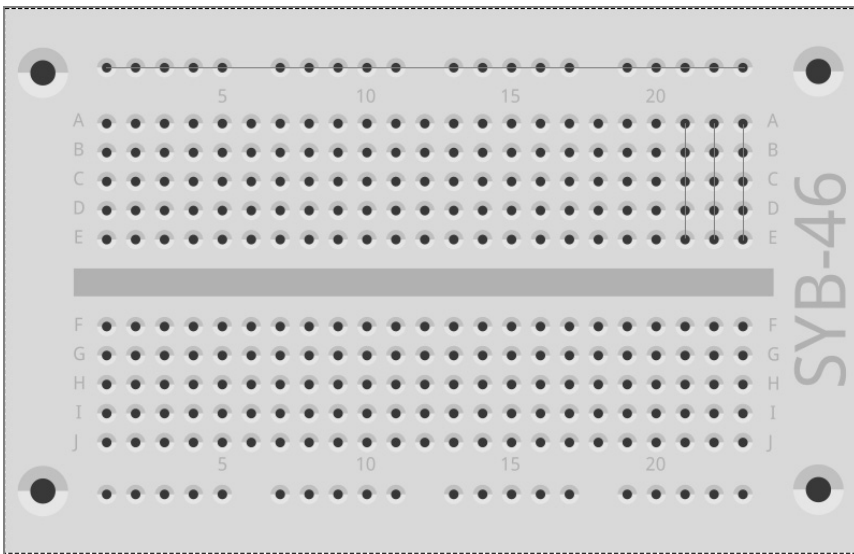
Het leerpakket bevat verschillende elektronische onderdelen, waarmee de beschreven experimenten (en natuurlijk ook eigen) kunnen worden opgebouwd. Op deze plaats worden de onderdelen alleen kort voorgesteld. De noodzakelijke praktijkervaring in het omgaan hiermee, zorgen dan voor de echte experimenten.

- 2x insteekprintplaat
- 1x LED rood
- 1x LED geel
- 1x LED groen
- 1x LED blauw
- 4x toetsen
- 4x weerstand 10 kilo-ohm (bruin-zwart-oranje)
- 4x weerstand 1 kilo-ohm (bruin-zwart-rood)

- 4x weerstand 220 kilo-ohm (rood-rood-bruin)
- 12x Verbindingskabel (insteekprintplaat - Raspberry Pi)
- ca. 1 m schakeldraad

### 2.1.1 Insteekprintplaten

Voor het snel opbouwen van elektronische schakelingen, zijn twee insteekprintplaten in het pakket opgenomen. Hier kunnen elektronische onderdelen direct op standaardafstanden in het gaatjesrooster worden gestoken, zonder te moeten solderen. Bij deze printplaten zijn de buitenste in de lengte geplaatste reeksen met contacten (X en Y) allemaal met elkaar verbonden.



Afb. 2.2: De insteekprintplaat uit het pakket met enkele, als voorbeeld bedoelde, aangegeven verbindingen.

Deze contactrijen worden vaak als plus- of minpool voor de voeding van de schakelingen gebruikt. In de overige contactrijen zijn steeds vijf contacten (A tot en met E en F tot en met J) dwars met elkaar verbonden, waarbij in het midden van de printplaat een ruimte is. Op deze manier kunnen hier in het midden grotere onderdelen worden ingestoken en naar buiten toe worden bedraad.

### 2.1.2 Verbindingskabels

De gekleurde verbindingskabels hebben allemaal aan een kant een kleine draadstekker, waarmee ze in de insteekprintplaat kunnen worden gestoken. Aan de andere kant bevindt zich een contactbus, die op een GPIO-pin van de Raspberry Pi past.

Er is bovendien staaldraad meegeleverd in het leerpakket. Hiermee maakt u korte verbingsbruggen, waarmee contactrijen op de insteekprintplaat worden verbonden. Knip het draad met een kleine zijkniptang op de passende lengtes af, zoals bij de individuele experimenten is beschreven. Om de draden beter in de insteekprintplaat te kunnen steken, wordt aanbevolen, de draden een beetje schuins af te snijden, zodat een soort wig ontstaat. Verwijder aan de beide uiteinden de isolatie op een lengte van ongeveer een halve centimeter.

### 2.1.3 Weerstanden en hun kleurcodes

Weerstanden worden in de digitale elektronica voor het overgrote deel voor stroombegrenzing aan de poorten van een microcontroller, alsmede als voorweerstanden voor LED's gebruikt. De maateenheid voor weerstanden is ohm. 1.000 ohm zijn één kilo-ohm, afgekort kilo-ohm.

De weerstandswaarden worden op de weerstanden weergegeven door gekleurde ringen. De meeste weerstanden hebben vier van dergelijke gekleurde ringen. De eerste beide gekleurde ringen geven de cijfers aan, de derde een multiplicator en de vierde de tolerantie. Deze tolerantiering is meestal goud- of zilverkleurig - dit zijn kleuren, die meestal niet voorkomen op de eerste ringen, zodat de leesrichting duidelijk is. De tolerantiewaarde speelt in de digitale elektronica nauwelijks een rol.

Kleur	Weerstandswaarde in Ohm			
	1. Ring (tien)	2. Ring (één)	3. Ring (multiplicator)	4. Ring (tolerantie)
Zilver			$10^{-2} = 0,01$	$\pm 10\%$
Goud			$10^{-1} = 0,1$	$\pm 5\%$
zwart		0	$10^0 = 1$	
Bruin	1	1	$10^1 = 10$	$\pm 1\%$
Rood	2	2	$10^2 = 100$	$\pm 2\%$
Oranje	3	3	$10^3 = 1.000$	
Geel	4	4	$10^4 = 10.000$	
Groen	5	5	$10^5 = 100.000$	$\pm 0,5\%$
Blauw	6	6	$10^6 = 1.000.000$	$\pm 0,25\%$
Paars	7	7	$10^7 = 10.000.000$	$\pm 0,1\%$
Grijs	8	8	$10^8 = 100.000.000$	$\pm 0,05\%$
Wit	9	9	$10^9 = 1.000.000.000$	

**Tab. 2.1:** De tabel toont de betekenis van de gekleurde ringen op weerstanden.

In het leerpakket zijn weerstanden in drie verschillende waarden opgenomen:

Waarde	1. Ring (tien)	2. Ring (één)	3. Ring (multipl.)	4. Ring (tolerantie )	Toepassing
220 Ohm	Rood	Rood	Bruin	Goud	Voorweerstand voor LED's
1 kilo-ohm	Bruin	zwart	Rood	Goud	Beveiligingsweerstand voor GPIO-ingangen
10 kilo-ohm	Bruin	zwart	Oranje	Goud	Pull-down-weerstand voor GPIO-ingangen

**Tab. 2.2:** Kleurcodes van de weerstanden in het leerpakket.

Let vooral bij de 1-kilo-ohm- en 10-kilo-ohm-weerstanden nauwkeurig op de kleuren. Deze kunnen gemakkelijk worden verwisseld.

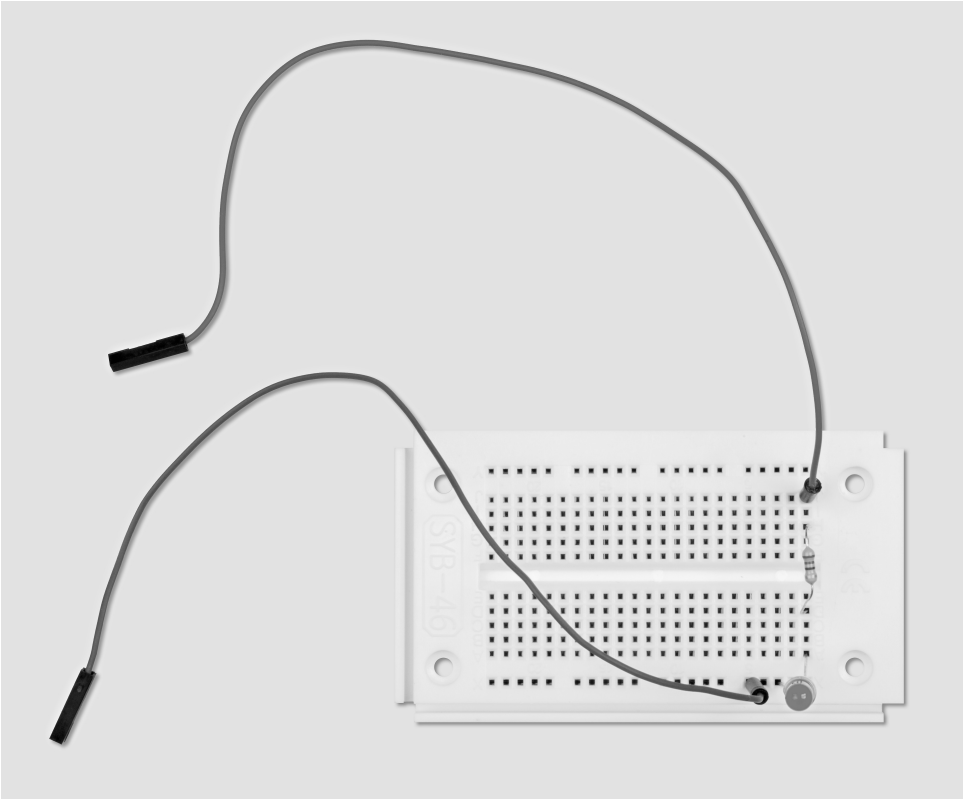
## 2.2 LED aansluiten

Aan de GPIO-poorten kunnen LED's (LED = Light Emitting Diode) worden aangesloten voor lichtsignalen en lichteffecten. Hierbij moet tussen de gebruikte GPIO-pin en de anode van de LED een 220-ohm-voorweerstand (rood-rood-bruin) worden ingebouwd, om de doorgevoerde stroom te begrenzen en hiermee doorbranden van de LED te voorkomen. Daarnaast beveiligt de voorweerstand tevens de GPIO-uitgang van de Raspberry Pi, omdat de LED in doorstroomrichting nauwelijks weerstand biedt en daarom de GPIO-poort bij verbinding met massa snel overbelast zou kunnen worden. De kathode van de LED verbindt men met de aardleiding op pin 6.

### In welke richting moet de LED aangesloten?

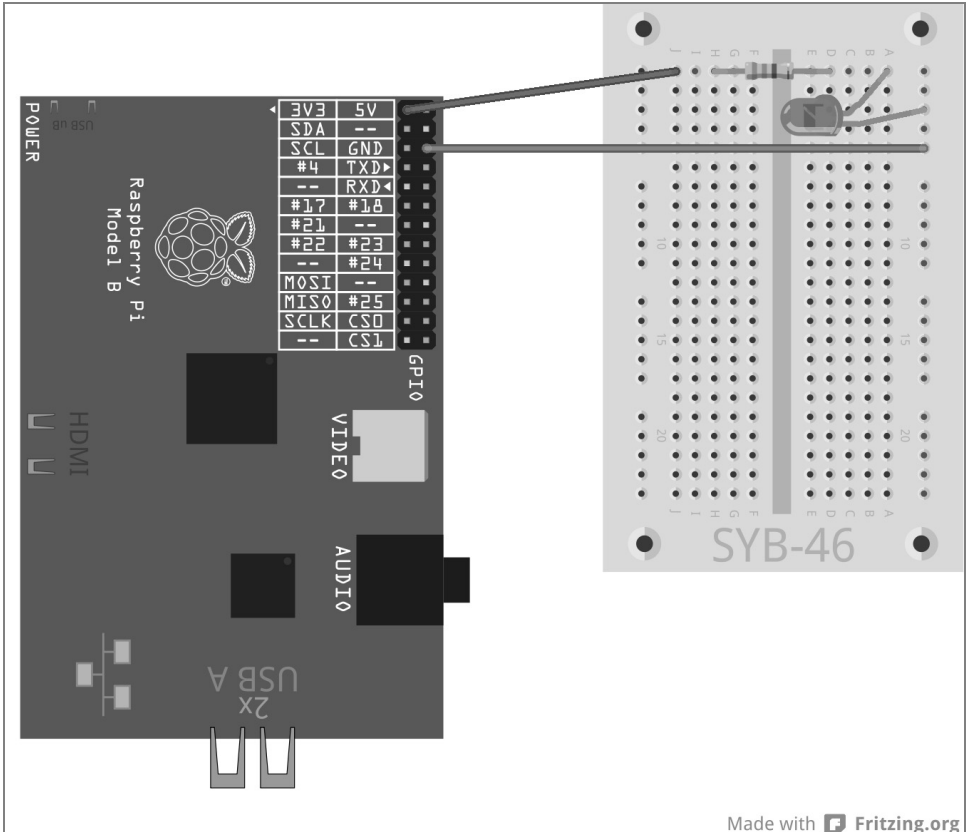
De beide aansluitdraden van een LED zijn verschillend van lengte. De langere van de twee is de pluspool, de anode, de kortere de kathode. Eenvoudig te onthouden: Het plusteken heeft één streep meer dan het minteken en maakt hiermee de draad iets langer. Bovendien zijn de meeste LED's aan de minkant afgevlakt, net als een minteken. Eenvoudig te onthouden: Kathode = kort = kant

Sluit als eerste een LED via een 220-ohm-voorweerstand (rood-rood-bruin) op een +3,3-V-aansluiting (pin 1) aan en verbind de minpool van de LED met de aardleiding (pin 6), zoals in de afbeelding weergegeven.



**Afb. 2.3:** Opbouw insteekbord, om een LED aan te sluiten.

Benodigde onderdelen:  
1x insteekprintplaat  
1x LED rood  
1x 220-ohm-weerstand  
2x verbindingskabel



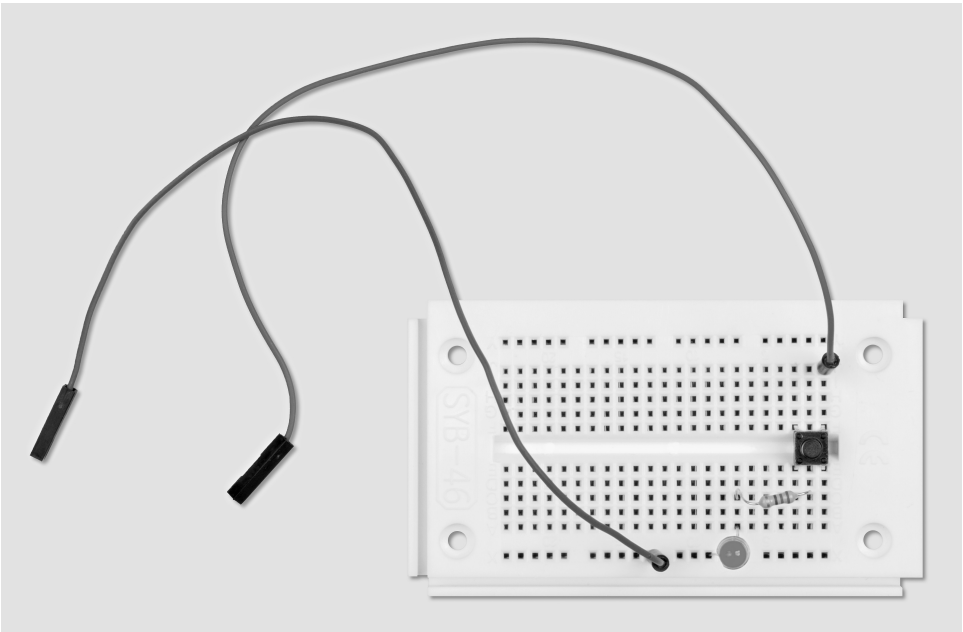
Made with Fritzing.org

**Afb. 2.4:** De eerste LED op de Raspberry Pi

In dit eerste experiment wordt de Raspberry Pi alleen gebruikt als voeding voor de LED. De LED brandt altijd, hiervoor is generlei software nodig.

In het volgende experiment plaatst u een toets in de aanvoerleiding van de LED. De LED brandt nu alleen, wanneer deze toets is ingedrukt. Ook hiervoor heeft men generlei software nodig.





**Afb. 2.5:** Opbouw insteekbord voor een LED, die met een toets wordt geschakeld.

**Benodigde onderdelen:**

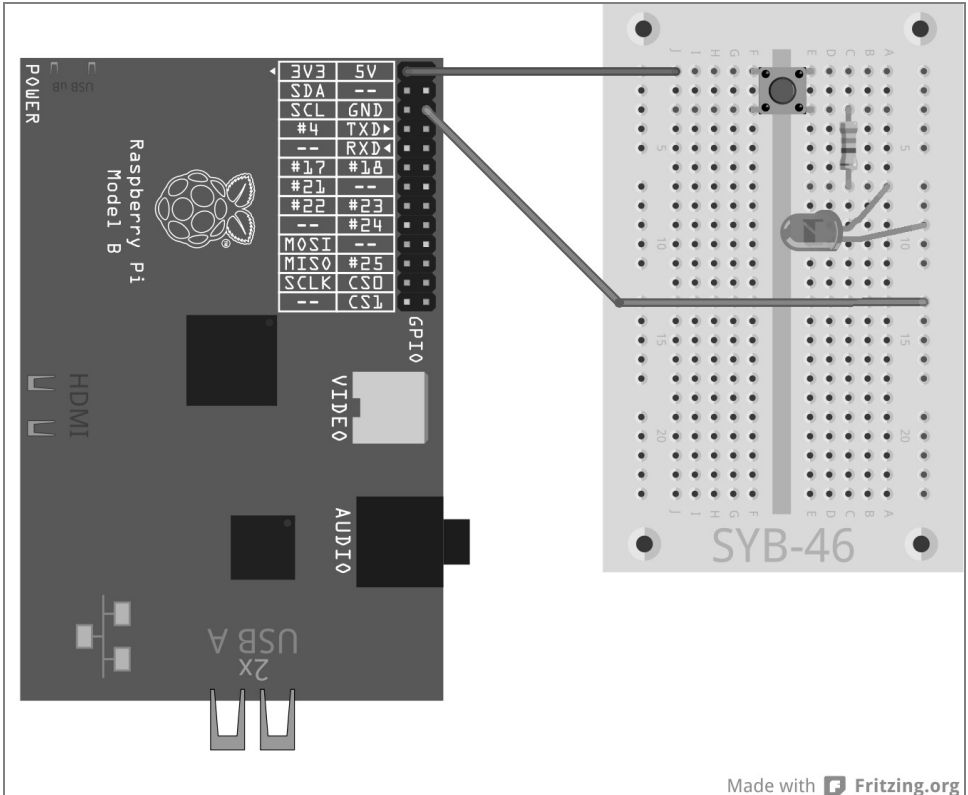
1x insteekprintplaat

1x LED rood

1x 220-ohm-weerstand

1x toets

2x verbindingskabel



Made with Fritzing.org

Afb. 2.6: LED met toets op de Raspberry Pi.

### 2.3 GPIO met Python

De Python-GPIO-bibliotheek moet zijn geïnstalleerd, om GPIO-poorten via Python-programma's te kunnen gebruiken. Twijfelt u, of alle noodzakelijke modules zijn geïnstalleerd, installeer een keer de actuele versies via de onderstaande console-opdrachten:

```

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

```

De GPIO-poorten zijn, net als onder Linux voor alle apparaten gebruikelijk, als bestanden in de mappenstructuur geïntegreerd. Om toegang te krijgen tot deze bestanden, heeft men root-rechten nodig. Dus start de Python-shell met root-rechten via een LXTerminal: `sudo idle`

## 2.4 LED aan- en uitschakelen

Sluit een LED via een 220-ohm-voorweerstand (rood-rood-bruin) aan op een GPIO-poort 25 (pin 22) en niet meer direct op een +3,3-V-aansluiting en verbind de minpool van de LED via de aardrail van de insteekprintplaat met de aardleiding van de Raspberry Pi (pin 6), zoals weergegeven in de onderstaande afbeelding.

Benodigde onderdelen:

1x insteekprintplaat

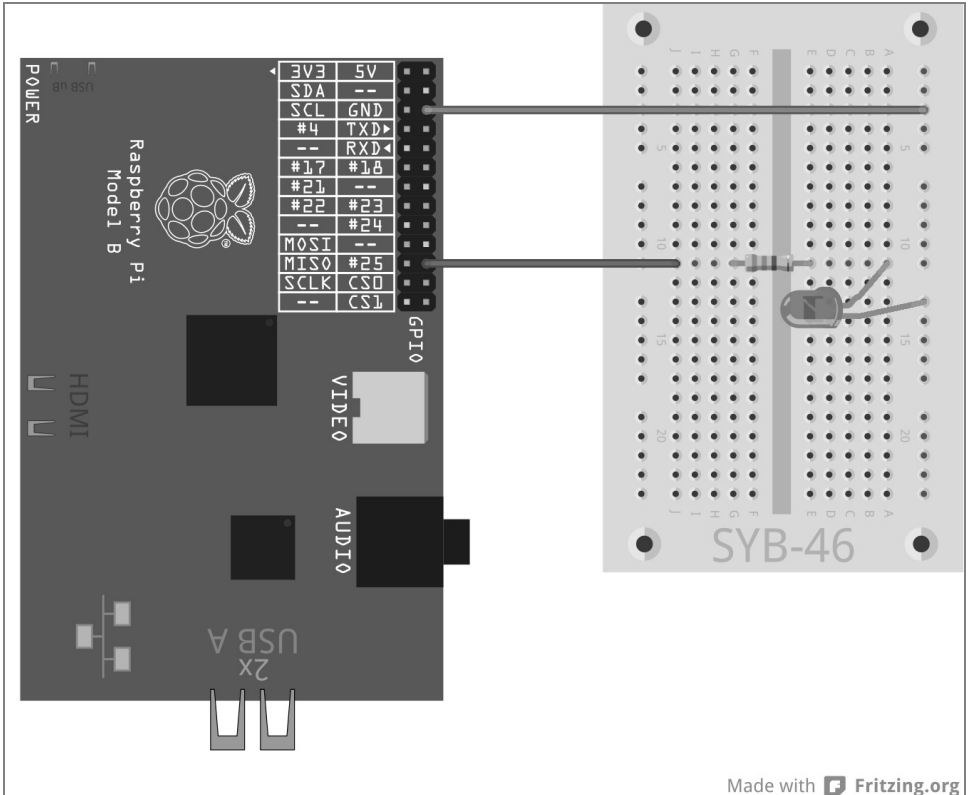
1x LED rood

1x 220-ohm-weerstand

2x verbindingkabel

Het volgende programma `led.py` schakelt de LED gedurende 5 seconden in en vervolgens weer uit:

```
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()
```



Afb. 2.7: Een LED aan de GPIO-poort 25.

### 2.4.1 Zo werkt het

Het voorbeeld toont de belangrijkste basisfuncties van de `RPi.GPIO`-bibliotheek.

`import RPi.GPIO as GPIO` De bibliotheek `RPi.GPIO` moet in elk Python-programma worden geïmporteerd, waarin ze zou moeten worden gebruikt. Door deze schrijfwijze kunnen alle functies van de bibliotheek via de prefix `GPIO` worden aangesproken.

`import time` De regelmatig gebruikte Python-bibliotheek `time` heeft niets te maken met de GPIO-programmering. Ze bevat functies voor de tijd- en datumberekening, onder andere ook een functie `time.sleep`, waarmee eenvoudig wachttijden in een programma kunnen worden gerealiseerd.

`GPIO.setmode(GPIO.BCM)` Aan het begin van elk programma moet worden gedefinieerd, hoe de GPIO-poorten zijn gekenmerkt. Doorgaans gebruikt men de standaardnummering `BCM`.

### Nummering van de GPIO-poorten

De bibliotheek `Rpi.GPIO` ondersteunt twee verschillende methoden voor het kenmerken van de poorten. In de modus `BCM` worden de bekende GPIO-poortnummers gebruikt, die ook op commandoregelniveau of in shell-scripts worden gebruikt. In de alternatieve modus `BOARD` komen de kenmerken van de pinnummers 1 tot en met 26 overeen met de Raspberry Pi-printplaat.

`GPIO.setup(25, GPIO.OUT)` De functie `GPIO.setup` initialiseert een GPIO-poort als uit- of als ingang. De eerste parameter kenmerkt de poort afhankelijk van de vooraf gegeven modus `BCM` of `BOARD` met zijn GPIO-nummer of pin-nummer. De tweede parameter kan of `GPIO.OUT` voor een uitgang of `GPIO.IN` voor een ingang zijn.

`GPIO.output(25, 1)` Op de zojuist geïntialiseerde poort verschijnt een 1. De hier aangesloten LED brandt. In plaats van de 1 kunnen ook de vooraf gedefinieerde waarden `True` of `GPIO.HIGH` verschijnen.

`time.sleep(5)` Deze functie uit de aan het begin van het programma geïmporteerde `time`-bibliotheek, zorgt voor een wachttijd van 5 seconden, voordat het programma verder loopt.

`GPIO.output(25, 0)` Om de LED uit te schakelen, voert men de waarde 0 resp. `False` of `GPIO.LOW` op de GPIO-poort uit .

`GPIO.cleanup()` Aan het eind van een programma moeten alle GPIO-poorten weer teruggezet worden. Deze regel verwerkt de voor alle door het programma geïntialiseerde GPIO-poorten in één keer. Poorten, die door andere programma's zijn geïntialiseerd, blijven ongewijzigd. Op deze manier wordt het verloop van deze andere, mogelijkkerwijze parallel lopende programma's, niet verstoord.

### GPIO-waarschuwingen ondervangen

Indien een GPIO-poort moet worden geconfigureerd, die niet volledig is teruggezet, echter waarschijnlijk door een ander of een afgebroken programma nog is geopend, ontstaan er waarschuwingen, die echter de programmastroom niet onderbreken. Deze waarschuwingen kunnen zeer nuttig zijn, om fouten te ontdekken tijdens de programmaontwikkeling. In een voltooid programma kunnen ze voor een onervaren gebruiker echter voor verwarring zorgen. Daarom biedt de GPIO-bibliotheek met `GPIO.setwarnings(False)` de mogelijkheid, deze waarschuwingen te onderdrukken.

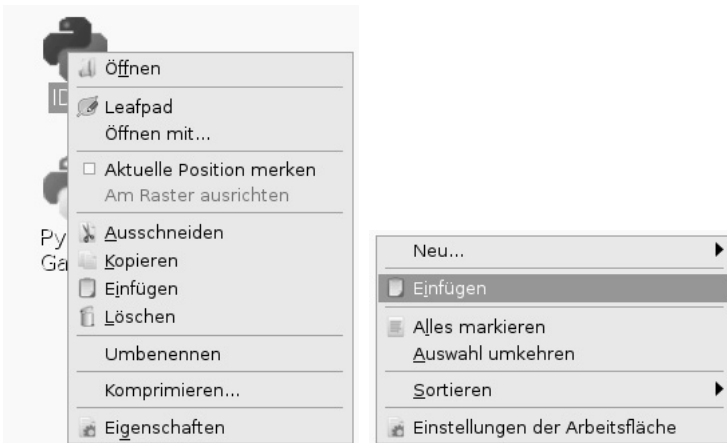
## 2.5 Python met GPIO-ondersteuning starten, zonder terminal

Wie veel met Python en GPIO bouwt, wil niet elke keer een LXTerminal opvragen, om IDLE te starten. Het kan ook eenvoudiger. Plaats hiervoor een pictogram op de desktop, die Python-IDE met superuser-rechten opvraagt:

- Maak een kopie van het vooraf geïnstalleerde desktoppictogram *IDLE*. Ga hiervoor als volgt te werk:



- Klik met de rechter muistoets op het pictogram *IDLE* op de desktop en kies in het contextmenu *kopiëren*.



Afb. 2.8: *IDLE*-desktoppictogram kopiëren.

Klik vervolgens met de rechter muistoets op de desktop en kies *Invoegen* in het contextmenu. Aangezien er al een gelijknamige desktopkoppeling bestaat, verschijnt bij de poging om een kopie te maken een melding.

Wijzig hier de naam van de kopie van *idle.desktop* in *idle\_gpio.desktop*. Aan het pictogram op de desktop wijzigt eerst niets. De weergegeven naam blijft *IDLE*.



Afb. 2.9: Melding bij het dupliceren van een desktopkoppeling.

Klik nu met de rechter muistoets op de kopie van het desktoppictogram en kies in het contextmenu *Leafpad*. Desktopkoppelingen zijn in Linux zuivere tekstbestanden, die met een tekstbewerker kunnen worden bewerkt.



Afb. 2.10: De desktopkoppeling in de Leafpad-Editor.

Voer hier de beide afgebeelde wijzigingen uit.

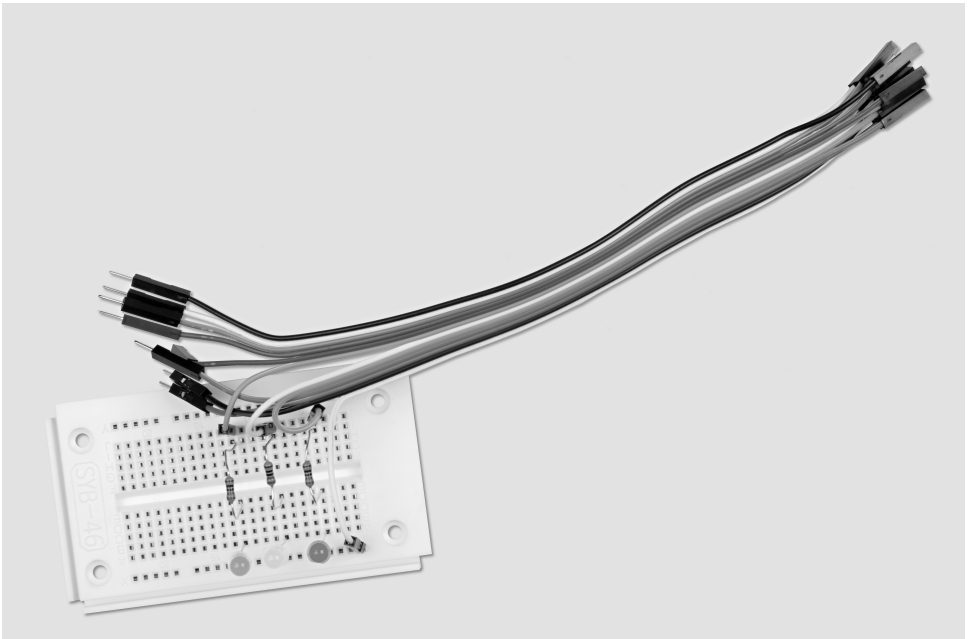
- Wijzig de veld naam naar IDLE GPIO. Dit is de op het beeldscherm weergegeven naam.
- Plaats in het veld Exec voor de eigenlijke opdrachtvraag het woord sudo.

Sluit de editor en sla het bestand op. Met een dubbelklik op het nieuwe desktoppictogram start u de Python-IDE *idle* met superuser-rechten. Nu kunt u de GPIO-functies gebruiken, zonder Python via een LXTerminal te moeten opvragen.

### 3 Verkeerslicht

Eén LED aan- en weer uitschakelen, kan in het eerste geval heel spannend zijn, maar hiervoor heeft men eigenlijk geen computer nodig. Een verkeerslicht met kenmerkende lichtcyclus van groen via geel naar rood en daarna via een lichtcombinatie rood-geel weer naar groen, is met drie LED's eenvoudig op te bouwen en toon overige programmeertechnieken in Python.

Bouw de afgebeelde schakeling op de insteekprintplaat. Voor de aansturing van de LED's worden drie GPIO-poorten en een gezamenlijke aardleiding gebruikt. De GPIO-poortnummers in de BCM-modus zijn op de Raspberry Pi in de tekening afgebeeld.

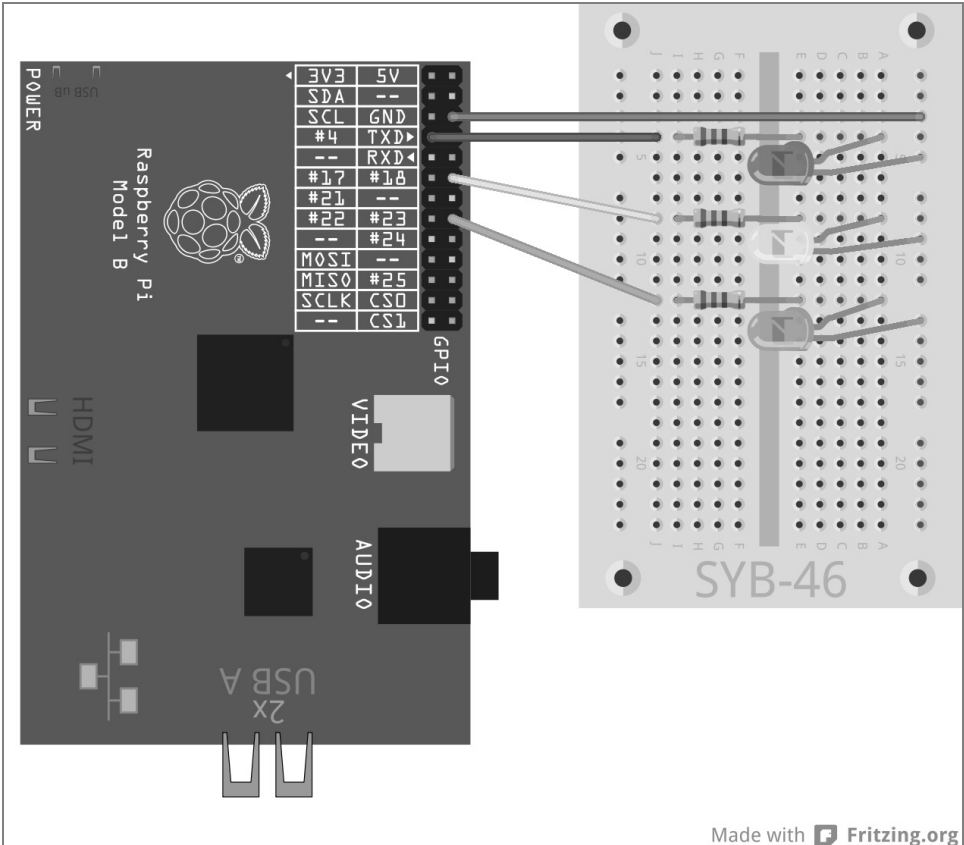


Afb. 3.1: Opbouw insteekbord voor het verkeerslicht.

Benodigde onderdelen:

- 1x insteekprintplaat
- 1x LED rood
- 1x LED geel
- 1x LED groen
- 3x 220-ohm-weerstand
- 4x verbindingkabel





Afb. 3.2: Een eenvoudig verkeerslicht.

Het programma `ampel01.py` sbestuur het stoplicht:

```
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 beëindigt het programma")
try:
    while True:
        time.sleep(2)
        GPIO.output(Ampel[gruen],False); GPIO.output(Ampel[gelb],True)
        time.sleep(0.6)
```

```

GPIO.output(Ampel[geel],False); GPIO.output(Ampel[rot],True)
time.sleep(2)
GPIO.output(Ampel[geel],True)
time.sleep(0.6)
GPIO.output(Ampel[rot],False); GPIO.output(Ampel[geel],False)
GPIO.output(Ampel[gruen],True)
except KeyboardInterrupt:
    GPIO.cleanup()

```

### 3.1.1 Zo werkt het

De eerste regels zijn al bekend, zij importeren de bibliotheken `RPi.GPIO` voor de aansturing van de GPIO-poorten en `time` voor tijdsvertragingen. Hierna wordt de nummering van de GPIO-poorten zoals in het eerdere voorbeeld op BCM geplaatst.

`rood = 0; geel = 1; groen = 2` Deze regels definiëren drie variabelen `rood`, `geel` en `groen` voor de drie LED's. Hierdoor hoeft men in het programma geen nummers of GPIO-poorten te onthouden, maar kan de LED's eenvoudig via hun kleuren aansturen.

`Ampel=[4,18,23]` Voor de aansturing van de drie LED's wordt een lijst ingesteld, die de GPIO-nummers in de volgorde bevat, waarin de LED's op de insteekprintplaat zijn gemonteerd. Omdat de GPIO-poorten alleen op deze ene plaats in het programma naar voren komen, kunt u het programma zeer eenvoudig ombouwen, wanneer u andere GPIO-poorten wilt gebruiken.

```

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

```

Achtereenvolgend worden de drie gebruikte GPIO-poorten als uitgangen geïnitieerd. Hierbij gebruiken wij dit keer geen GPIO-poortnummers, maar de eerder gedefinieerde lijst. Binnen een lijst worden de individuele elementen via getallen, beginnend met 0, geïndiceerd. `Ampel[0]` is dus het eerste element, in dit geval 4. De variabelen `rood`, `geel` en `groen` bevatten de getallen 0, 1 en 2, die als indicaties voor de elementen van de lijst nodig zijn. Op deze manier kunnen de gebruikte GPIO-poorten via kleuren worden geadresseerd:

- `Ampel[rot]` komt overeen met de GPIO-poort 4 met de rode LED.
- `Ampel[geel]` komt overeen met de GPIO-poort 18 met de gele LED.
- `Ampel[gruen]` komt overeen met de GPIO-poort 23 met de groene LED.

De `GPIO.setup`-aanwijzing kan een optionele parameter `initial` bevatten, die aan de GPIO-poort bij het initialiseren al een logische status toewijst. Hiermee schakelen we in dit programma de groene LED al vanaf het begin in. De andere beide LED's beginnen het programma in uitgeschakelde toestand.

`print("Strg+C beëindigt het programma")` Nu verschijnt een korte gebruiksaanwijzing op het beeldscherm. Het programma loopt automatisch. De toetscombinatie `[Strg]+[C]` moet het beëindigen. Om te vragen, of de gebruiker met `[Strg]+[C]` het programma beëindigt, gebruiken wij een `try...except`-vraag. Hierbij wordt de onder `try`: ingevoerde programmacode vervolgens normaal uitgevoerd. Wanneer gedurende deze een systeemuitzondering optreedt - dit kan een fout zijn of mede de toetscombinatie

`Strg` + `C` -, wordt het programma afgebroken en de `except`-aanwijzing aan het einde van het programma wordt uitgevoerd..

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

Door deze toetscombinatie wordt een `KeyboardInterrupt` geactiveerd en de lus wordt automatisch verlaten. De laatste regel sluit de gebruikte GPIO-poorten en schakelt hiermee alle LED's uit. Hierna wordt het programma beëindigd. Door het gecontroleerde sluiten van de GPIO-poorten komen geen systeemwaarschuwingen of afbrek meldingen naar voren, waardoor de gebruiker in de war kan raken. De eigenlijke verkeerslichtcyclus loopt in een gesloten lus:

`while True` : Dergelijke gesloten lussen hebben altijd een afbreekvoorwaarde nodig, omdat het programma anders nooit zou worden beëindigd.

`time.sleep(2)` Aan het begin van het programma en tevens bij elk nieuw begin van de lus brandt de groene LED gedurende 2 seconden.

```
GPIO.output(Ampel[gruen],False); GPIO.output(Ampel[ge1b],True)  
time.sleep(0.6)
```

Nu wordt de groene LED uit- en hiervoor de gele LED ingeschakeld. Deze brandt dan gedurende 0,6 seconden alleen.

```
GPIO.output(Ampel[ge1b],False); GPIO.output(Ampel[rot],True)  
time.sleep(2)
```

Nu wordt de gele LED weer uit- en hiervoor de rode LED ingeschakeld. Deze brandt dan gedurende 2 seconden alleen. De roodfase van een verkeerslicht is gewoonlijk duidelijk langer dan de geelfase.

```
GPIO.output(Ampel[ge1b],True)  
time.sleep(0.6)
```

Voor de start van de rood-geel-fase wordt de gele LED aanvullend ingeschakeld, zonder dat een andere LED wordt uitgeschakeld. Deze fase duurt 0,6 seconden.

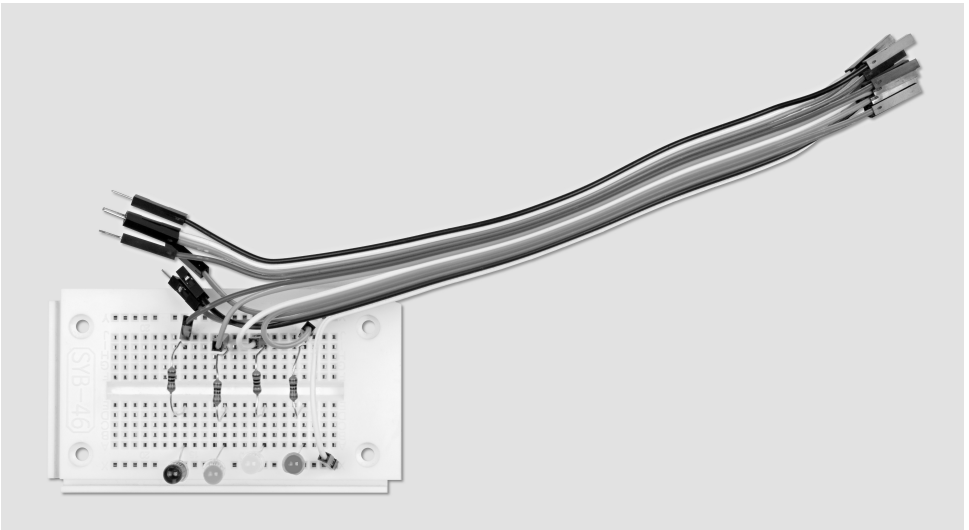
```
GPIO.output(Ampel[rot],False)  
GPIO.output(Ampel[ge1b],False)  
GPIO.output(Ampel[gruen],True)
```

Aan het einde van de lus springt het verkeerslicht weer op groen. De rode en gele LED worden uitgeschakeld, de groene wordt ingeschakeld. De lus begint in de groenfase van het verkeerslicht opnieuw met een wachttijd van 2 seconden. U kunt natuurlijk alle tijden willekeurig aanpassen. In het echt zijn de verkeerslichtfasen afhankelijk van de afmetingen van de kruising en de verkeersstromen. De geel- en rood-geel-fase duren doorgaans elk 2 seconden.

## 4 Voetgangerslicht

In het volgende experiment breiden we de verkeerslichtschakeling nog met een extra voetgangerslicht uit, die tijdens de rodfase van het verkeerslicht een knipperlicht voor voetgangers weergeeft, zoals het in sommige landen wordt gebruikt. Men kan natuurlijk ook het in Midden-Europa gebruikelijke voetgangerslicht met rood en groen licht in het programma inbouwen, alleen bevat dit leerpakket naast de voor het verkeerslicht gebruikte LED's, nog slechts één andere.

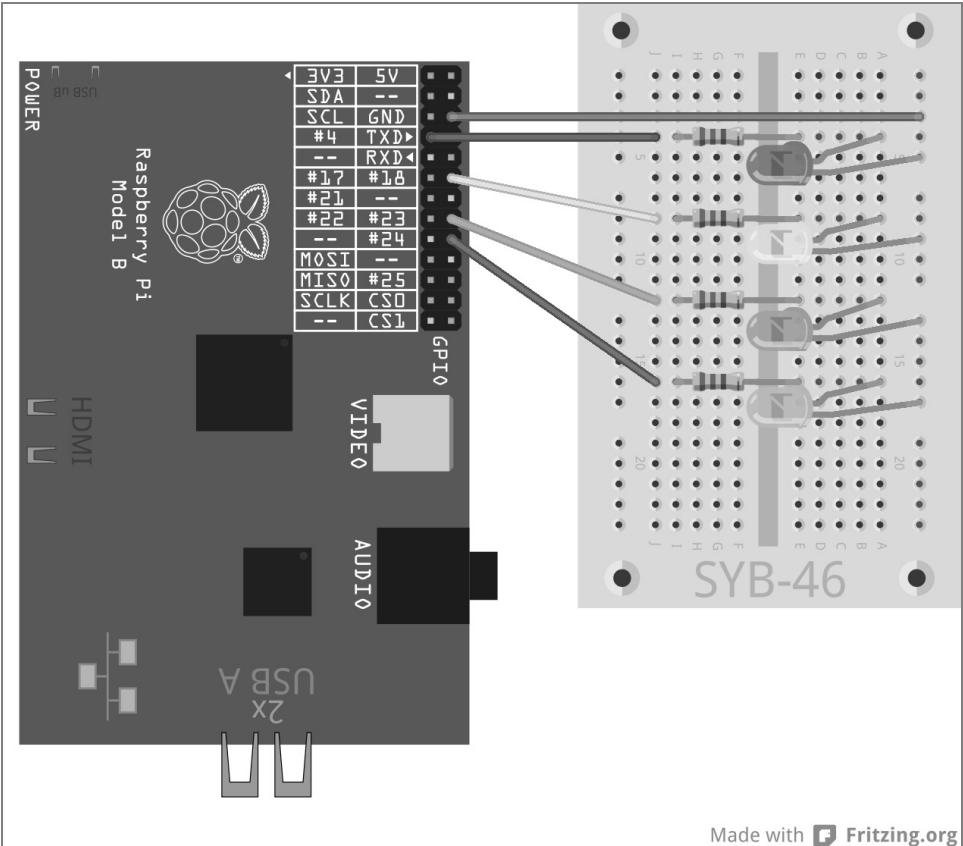
Bouw voor het volgende experiment een extra LED met voorweerstand in, zoals afgebeeld in de schakeling. Deze wordt aangesloten op de GPIO-poort 24.



Afb. 4.1: Opbouw insteekbord voor verkeerslicht en voetgangerlicht

### Benodigde onderdelen:

- 1x insteekprintplaat
- 1x LED rood
- 1x LED geel
- 1x LED groen
- 1x LED blauw
- 4x 220-ohm-weerstand
- 5x verbindingkabel



Made with Fritzing.org

Afb. 4.2: Verkeerslicht met voetgangerlicht.

Het programma `ampel01.py` bestuurt het nieuwe verkeerslicht: Ten opzicht van de eerdere versie is het programma maar gering uitgebreid.

```
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 beëindigt het programma")
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 Zo werkt het

Het programmaverloop is uitgebreid bekend. Tijdens de nu iets langere roodfase moet het blauwe voetgangerslicht snel knipperen.

`blauw = 4` Een nieuwe variabele definieert de LED voor het voetgangerslicht in de lijst.

`Ampel=[4,18,23,24]` De lijst wordt met vier elementen uitgebreid, om de vier LED's te kunnen aansturen.

`GPIO.setup(Ampel[blau], GPIO.OUT, initial=False)` De nieuwe LED wordt geïnitieerd en aanvankelijk uitgeschakeld. Dit is de basisinstelling tijdens de groenfase van het verkeerslicht.

```

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)

```

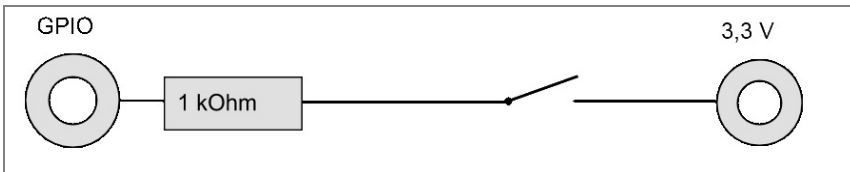
In de verkeerslichtcyclus start 0,6 seconden na begin van de roodfase een lus, die de blauwe LED laat knipperen. Hiervoor gebruiken wij hier een `for`-lus, die in tegenstelling tot de in de voorgaande experimenten gebruikte `while`-lussen altijd een bepaald aantal lusdoorlopen gebruikt, en niet loopt tot aan een bepaalde afbreekvoorwaarde is voldaan.

`for i in range(10):` Elke `for`-lus heeft een lusteller nodig, een variabele, die bij elke lusdoorloop een nieuwe waarde aanneemt. Voor eenvoudige lustellers is in alle programmeertalen de variabelennaam `i` ingeburgerd. Elke andere naam is natuurlijk ook mogelijk. Deze waarde kan als elke andere variabele binnen de lus worden opgevraagd, wat hier echter niet nodig is. De parameter `range` in de lus geeft aan, hoe vaak de lus doorloopt, nauwkeuriger gezegd, welke waarden de lusteller kan aannemen. In ons voorbeeld loopt de lus tien keer. De lusteller `i` krijgt hierbij waarden van 0 tot en met 9. Binnen de lus wordt de nieuwe blauwe LED ingeschakeld en na 0,05 seconden weer uitgeschakeld. Na meerdere 0,05 seconden is een lusdoorloop beëindigd en de volgende start weer met het inschakelen van de LED. Op deze manier knippert ze tien keer, dit duurt in totaal 1 seconde.

`time.sleep(0.6)` Met een vertraging van 0,6 seconden na de laatste lusdoorloop wordt de normale schakelcyclus van het verkeerslicht voortgezet, doordat de gele LED aanvullend op de al brandende rode wordt ingeschakeld. Tot nu toe niet veel nieuws. Het voetgangerslicht wordt pas echt interessant, wanneer ze niet automatisch loopt, maar pas door een toetsindruk wordt gestart, zoals die bij veel voetgangerslichten het geval is. In het volgende experiment wordt een op een GPIO-poort aangesloten toets de drukknop aan een echt voetgangerslicht simuleren.

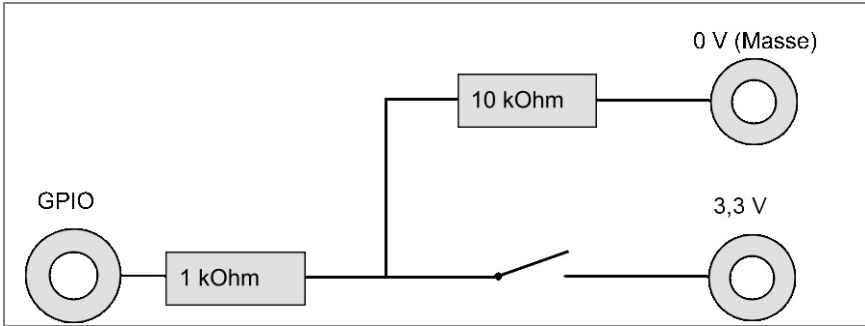
## 4.2 Toets op de GPIO-aansluiting

GPIO-poorten kunnen niet alleen gegevens uitvoeren, bijvoorbeeld via LED's, maar ook voor gegevensinvoer worden gebruikt. Hiervoor moeten ze in het programma als ingang worden gedefinieerd. Voor de invoer gebruiken we in het volgende project een toets, die direct in de insteekprintplaat wordt gestoken. De toets heeft vier aansluitpinnen, waarbij per twee tegenoverliggende (grote afstand) met elkaar zijn verbonden. Zolang de toets is ingedrukt, zijn alle vier aansluitingen met elkaar verbonden. In tegenstelling tot een schakelaar klikt een toets niet vast. De verbinding wordt bij het loslaten direct weer losgekoppeld. Als op een als ingang gedefinieerde GPIO-poort een +3,3-V-sigitaal aanwezig is, wordt deze als logisch `True` resp. `1` beoordeeld. Theoretisch kunt u dus via één toets de betreffende GPIO-poort met de +3,3-V-aansluiting van de Raspberry Pi verbinden, dit mag u echter absoluut niet doen! De GPIO-poort wordt hierdoor overbelast. Sluit altijd een 1-kilo-ohm-beveiligingsweerstand aan tussen een GPIO-ingang en de +3,3-V-aansluiting, waardoor veel stroom op de GPIO-poort en dus op de processor stroomt.



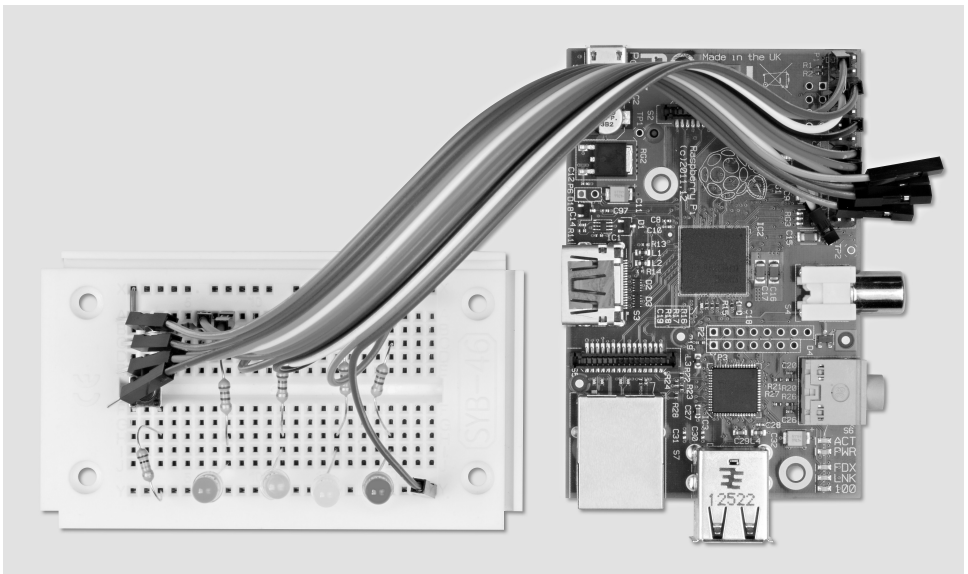
Afb. 4.3: Toetsen met beveiligingsweerstand aan een GPIO-ingang.

In de meeste gevallen werkt deze eenvoudige schakeling al, echter had de GPIO-poort bij open toets geen duidelijk gedefinieerde toestand. Wanneer een programma deze poort oproept, kan dit in willekeurige resultaten resulteren. Om dit te voorkomen, sluit men een vergelijkbare zeer hoge weerstand aan - doorgaans 10 kilo-ohm - tegen aarding. Deze zogenaamde pull-down-weerstand trekt de status van de GPIO-poorten bij geopende toets weer naar beneden naar 0 V. Omdat de weerstand zeer hoog is, bestaat, zolang de toets wordt ingedrukt, ook geen kortsluitingsgevaar. Wanneer de toets is ingedrukt is +3,3 V en de aardleiding direct via deze weerstand verbonden.



**Afb. 4.4:** Toets met beveiligingsweerstand en pull-down-weerstand aan een GPIO-ingang.

Bouw volgens de onderstaande afbeelding een toets met de beide weerstanden in.



**Afb. 4.5:** Opbouw insteekbord voor de voetganger-»bedieningslicht«.

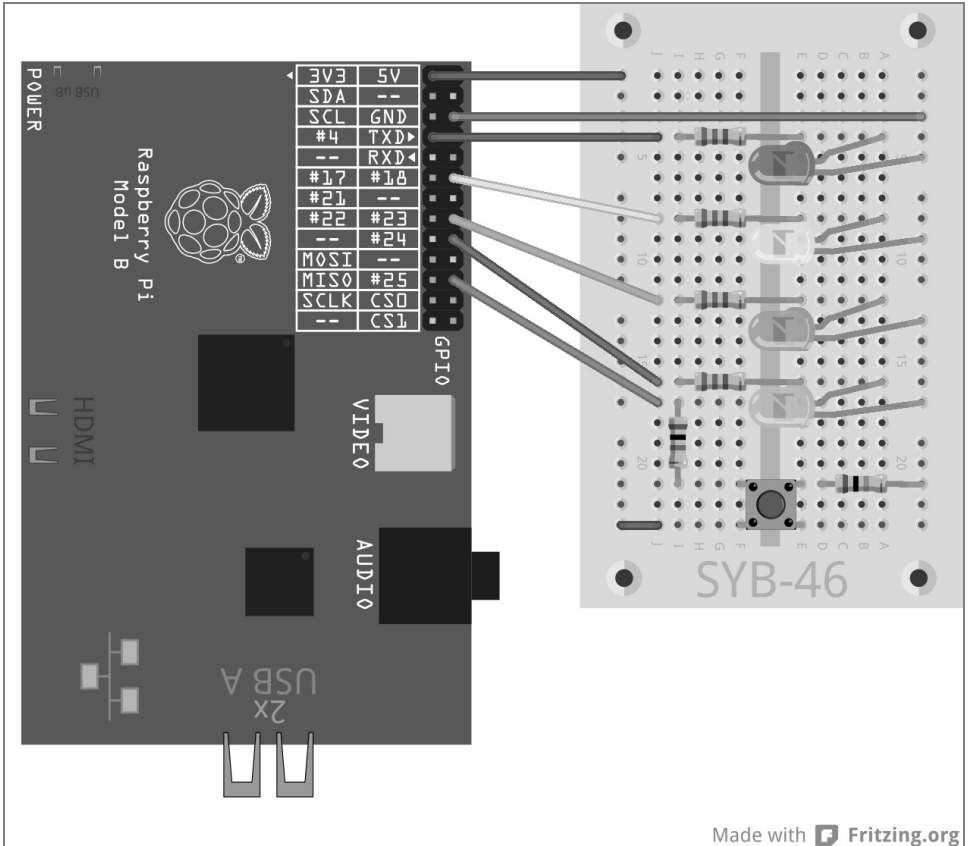


**Benodigde onderdelen:**

1x insteekprintplaat  
1x LED rood  
1x LED geel  
1x LED groen  
1x LED blauw  
4x 220-ohm-weerstand  
1x toets  
1x 1-kilo-ohm-weerstand  
1x 10-kilo-ohm-weerstand  
7x verbindingkabel  
1x korte draadbrug

De in de afbeelding weergegeven onderste contactstrip van de toets is via de plus-rail van de insteekprintplaat met de +3,3-V-leiding van de Raspberry Pi (pin 1) verbonden. Voor de verbinding van de toets met de plus-rail gebruiken wij, om de tekening overzichtelijk te houden, een korte draadbrug. Eventueel kunt u ook een van de onderste contacten van de toets direct met een verbindingkabel met de pin 1 van de Raspberry Pi verbinden.

De in de afbeelding weergegeven contactstrip van de toets is via een 1-kilo-ohm-beveiligingsweerstand (bruin-zwart-rood) met de GPIO-poort 25 verbonden en via een 10-kilo-ohm-pull-down-weerstand (bruin-zwart-oranje) met de aardleiding.



Made with Fritzing.org

Afb. 4.6: Voetgangersknipperlicht met toets

Het programma `ampel103.py` bestuurt de nieuwe verkeerslichtinstallatie met de toets voor het voetgangersknipperlicht.

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

GPIO.setmode(GPIO.BCM)
rood = 0; geel = 1; groen = 2; blauw = 3; toets = 4

Ampel=[4,18,23,24,25]
GPIO.setup(Ampel[rood], GPIO.OUT, initial=False)
GPIO.setup(Ampel[geel], GPIO.OUT, initial=False)
GPIO.setup(Ampel[groen], GPIO.OUT, initial=True)
GPIO.setup(Ampel[blauw], GPIO.OUT, initial=False)
```

```

GPIO.setup(Ampel[taster], GPIO.IN)

print ("Druk de toets in, om het voetgangerlicht in te schakelen, Strg+C beëindigt het
programma")

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

```

#### 4.2.1 Zo werkt het

Het programma is ten opzichte van de laatste versie nog enigszins aangevuld.

# -\*- coding: utf-8 -\*- **Opdat de Nederlandse dubbele punt van het voetgangersknipperlicht in de programma-uitvoer juist worden weergegeven - onafhankelijk van hoe de IDLE-interface is ingesteld bij de gebruiker -, wordt aan het begin een codering voor de aanduiding van de speciale tekens gedefinieerd. Deze regel moet in alle programma's zijn opgenomen, de teksten uitvoeren, waarin zich dubbele punten of andere landspecifieke speciale tekens bevinden.**

#### ASCII, ANSI en Unicode

Een normaal alfabet heeft 26 letter plus een paar dubbele punten, allemaal in hoofd- en kleine letters, daarnaast 10 cijfers en enkele leestekens; gezamenlijk ongeveer 100 verschillende tekens. Eén Byte kan 256 verschillende tekens weergeven. Dit zou dus toereikend moeten zijn - dat dacht men aan het begin van het computertijdperk, toen de belangrijkste basis voor de huidige techniek werden gedefinieerd.

Al snel ontdekte men, dat de uitvinders van de op 256 tekens gebaseerde ASCII-leestekens (American Standard Code for Information Interchange) het fout hadden. Het waren Amerikanen, die niet verder hadden gedacht dan de Engelse taal. In alle belangrijke wereldtalen, zonder de Oost-Aziatische en Arabische talen met hun volledig eigen schrift, bestaan er een veelvoud van honderd letters die moeten worden weergegeven. Hiervan pasten slechts weinig op de vrije plaatsen in de 256 tekens omvattende lijst.

Toen later parallel aan de ASCII-leestekens de ANSI-leestekens werden ingevoerd, die door oudere Windows-versies wordt gebruikt, maakte men dezelfde fout nog een keer. Om de taalverwarring perfect te maken, werden de Duitse dubbel punten en andere letters met accenten op andere plaatsen in de leestekens geplaatst dan in de ASCII-standaard.

Om dit probleem op te lossen voerde men in de jaren 90 de Unicode in, die alle talen, waaronder tevens Egyptische hiëroglfen, spijkerschrift en het Vedisch Sanskriet, de oudste in de wereld overgeleverde geschreven taal, kan weergeven. De meest verbreide vorm, Unicode-tekenen in zuivere tekstbestanden te coderen, is UTF-8, een codering, die platformoverkoepelend werkt en de eerste 128 tekens met ASCII identiek - en dus neerwaarts compatibel is met bijna alle tekstweergevende systemen. De codering wordt in een commentaarregel aangegeven. Alle regels die met een #-teken beginnen, worden niet door de Python-interpretor beoordeeld. De codering, die altijd helemaal aan het begin van het programma moet staan, laat de Python-shell zien, hoe tekens moeten worden weergegeven, is echter geen echte programma-aanwijzing. Op deze manier kunt u ook willekeurige eigen commentaren in de programmacode invoeren.

### **Commentaren in programma's**

Tijdens het schrijven van een programma, weet men later vaak niet meer, wat men van plan was bij bepaalde programma-aanwijzingen. Programmeren is een van de meest creatieve bezigheden, omdat men enkel en alleen, zonder beperkingen door materiaal en hulpmiddel, vanuit ideeën iets creëert. Juist bij programma's, die ook een andere persoon begrijpt of zelfs verder moet bewerken, zijn commentaren belangrijk. In het voorbeeldprogramma zijn geen commentaren opgenomen, om de programmacode overzichtelijk te houden. Alle programma-aanwijzingen zijn uitgebreid beschreven.

Bij programma's, die men zelf publiceert, vraagt men zich altijd af: commentaren in Nederlands of Engels? Bij Nederlandse commentaren klagen de Fransen over de onbegrijpelijke taal, zelfs de Engelse taal begrijpt men op een gegeven moment niet meer, en de Britten lachen vanwege het slechte Engels.

```
toets = 4
verkeerslicht=[4,18,23,24,25]
```

Omwille van de eenvoud wordt voor de toets een extra element met het nummer 4 en de GPIO-poort 25 aan de lijst toegevoegd. Op deze manier kan men voor de toets ook gemakkelijk een andere GPIO-poort kiezen, omdat de nummers hiervan net als de GPIO-poorten van de LED's alleen op deze plaats in het programma is ingevoerd.

`GPIO.setup(Ampel[taster], GPIO.IN)` De GPIO-poort van de toets wordt als ingang gedefinieerd. Deze definitie gebeurt tevens via `GPIO.setup`, dit keer echter met de parameter `GPIO.IN`.

```
print("Druk op de toets, om het voetgangersknipperlicht in te schakelen, Strg+C beëindigt het programma")
```

Tijdens de start geeft het programma nog een andere melding weer, die mededeelt, dat de gebruiker de toets moet indrukken.

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

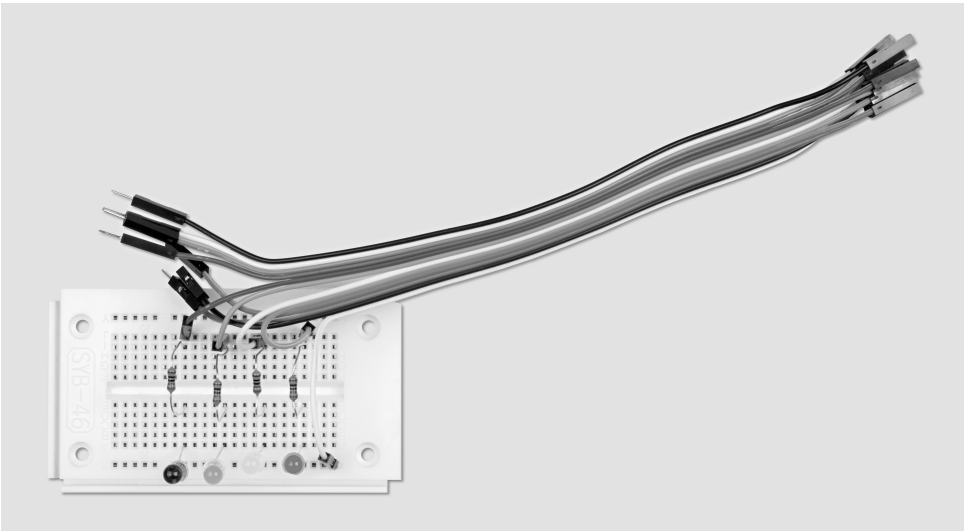
Binnen de gesloten lus is nu een vraag ingebouwd. De onderstaande aanwijzingen worden pas uitgevoerd, als de GPIO-poort 25 de waarde `True` aanneemt, de gebruiker dus een toets indrukt. Tot aan dit moment blijft het verkeerslicht in haar groenfase staan. Het verdere verloop van de lus komt in wezen overeen met die van het laatste programma. Het verkeerslicht schakelt via geel naar rood, het knipperlicht knippert tien keer. Hierna schakelt het verkeerslicht weer via rood met geel naar groen.

`time.sleep(2)` In dit programma is sprake van een klein verschil. De 2 seconden durende groenfase is nu aan het einde van de lus en niet meer aan het begin ingebouwd. Toch wordt ze één keer per lusdoorloop toegepast, met het verschil, dat de verkeerslichtcyclus onmiddellijk en zonder vertraging begint, wanneer de toets wordt ingedrukt. Om te voorkomen, dat de groenfase bijna uitvalt, wanneer de toets direct na de geelfase opnieuw wordt ingedrukt, is deze vertraging nu aan het einde van de lus ingebouwd.

## 5 Gekleurde LED-patronen en looplichten

Looplichten zijn altijd weer geliefde effecten om aandacht te krijgen, hetzij in de feestkelder of in professionele lichtreclame. Met de Raspberry Pi en een paar LED's kan zo iets gemakkelijk worden gerealiseerd.

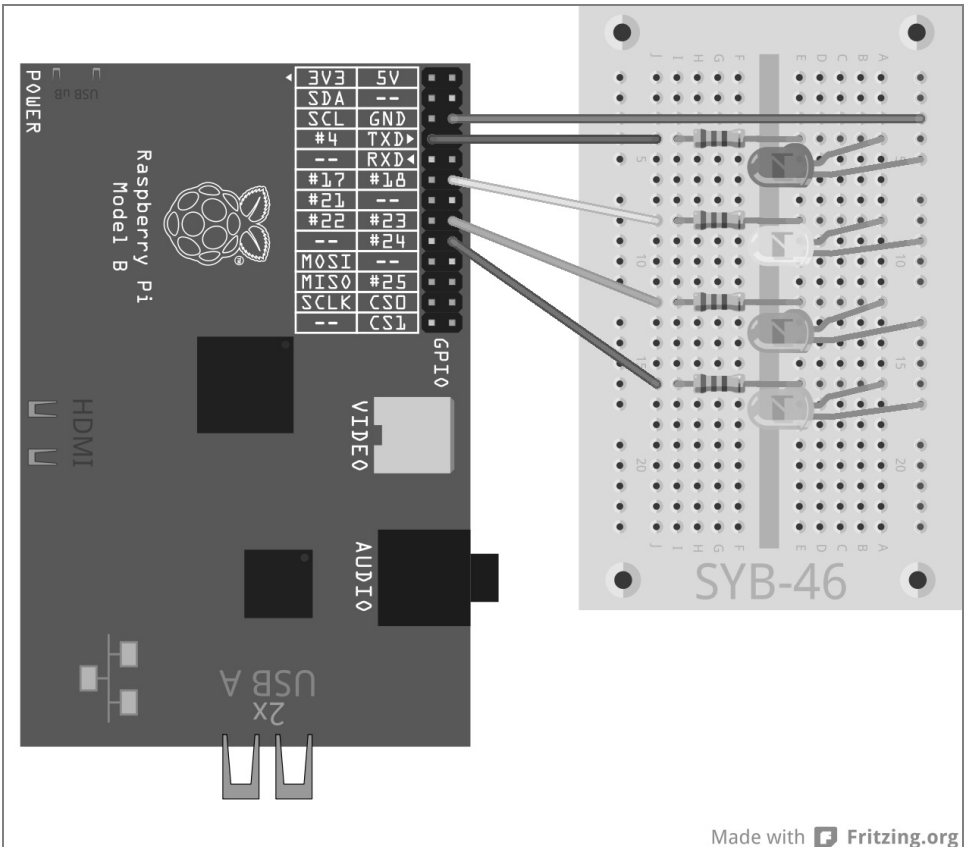
Bouw voor het volgende experiment vier LED's met voorweerstand, zoals afgebeeld. Deze schakeling komt overeen met het voetgangerslicht zonder de toets uit het vorige experiment.



Afb. 5.1: Opbouw insteekbord voor de patronen en looplichten.

Benodigde onderdelen:

- 1x insteekprintplaat
- 1x LED rood
- 1x LED geel
- 1x LED groen
- 1x LED blauw
- 4x 220-ohm-weerstand
- 5x verbindingskabel



Afb. 5.2: Vier LED's met voorweerstand.

Aan de hand van verschillende LED-knipperpatronen, leggen wij andere lussen en programmeermethoden uit in Python. Het volgende programma biedt verschillende LED-patronen, die door de gebruiker via toetsenbordinput kan worden gekozen.

Het programma `ledmuster.py` laat de LED's in verschillende patronen knipperen.

```
# -*- 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 ("keuze in lichteffecten"); print ("1 - looplicht cyclisch")
print ("2 - looplicht heen en weer"); print ("3 - op- en neergaand")
print ("4 - allen knipperen gelijktijdig"); print ("5 - allen knipperen toevallig")
print ("Strg+C beëindigt het programma")

try:
    while True:
        e = raw_input ("Kies een voorbeeld a.u.b.: ")
        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 ("Ongeldige invoer")

except KeyboardInterrupt:
    GPIO.cleanup()

```

### 5.1.1 Zo werkt het

De eerste regels van het programma met de definitie van de UTF-8-codering en de import van de noodzakelijke bibliotheken zijn al vanuit eerdere experimenten bekend. Hier wordt vervolgens de bibliotheek `random` geïmporteerd, om een willekeurig knipperpatroon te creëren.

Bij de ontwikkeling van dit programma is er rekening mee gehouden, dat het multi-inzetbaar is, zich dus probleemloos met meer dan vier LED's kan worden uitgebreid. Bij een goede programmeerstijl hoort tegenwoordig een dergelijke flexibiliteit. In het voorbeeld van de Raspberry Pi kunnen dergelijke geprogrammeerde programma's niet alleen met nieuwe GPIO-poorten worden uitgebreid, maar ook gemakkelijk op andere GPIO-poorten overschrijven, mocht dit hardwaretechnisch noodzakelijk zijn.

`LED = [4,18,23,24]` Voor de LED's wordt weer een lijst met GPIO-nummers gedefinieerd, opdat men deze poorten alleen vast moet invoeren op een plaats in het programma.

```

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

```

In plaats van, zoals in vorige programma's, de GPIO-poorten van de LED's apart te initialiseren, loopt dit keer een `for`-lus via de lijst `LED`. De lusteller `i` neemt achtereenvolgens de individuele waarden uit de lijst aan, in het voorbeeld de GPIO-poortnummer van de LED's en wordt niet eenvoudig bijgeteld zoals in de gebruikte `for`-lussen tot nu toe. Op deze manier kunnen willekeurig lange lijsten worden afgewerkt. De lengte van de lijst hoeft tijdens de ontwikkeling van het programma niet eens bekend te zijn.

De vier GPIO-poorten voor de LED's worden als uitgangen gedefinieerd en op 0 ingesteld, om eventueel uit eerdere experimenten brandende LED's uit te schakelen.

```

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

```



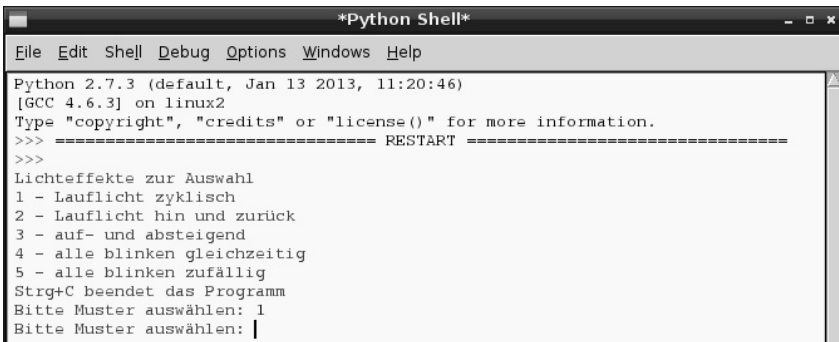
Om het programma algemeen geldend en gemakkelijk wijzigbaar te houden, worden nu nog slechts drie variabelen gedefinieerd:

z	Aantal LED's	Het aantal LED's wordt met behulp van de functie <code>len()</code> automatisch uit de lijst LED overgenomen
w	Herhalingen	Elk patroon wordt, opdat men het beter herkent, standaard vijf keer herhaald. Dit aantal kan willekeurig worden gewijzigd en geldt dan voor alle patronen.
t	Tijd	Deze variabele geeft weer, hoe lang een LED tijdens het knipperen is ingeschakeld. De hierop volgende pauze duurt overeenkomstig lang. De naam <code>t</code> is voor variabelen, die tijdstippen in programma's opslaan, in bijna alle programmeertalen ingeburgerd.

De als variabelen gedefinieerde waarden zijn alleen op deze plaats vast in het programma ingebouwd en kunnen zo gemakkelijk worden gewijzigd. Na deze definities start het eigenlijke programma.

```
print ("Keuze uit lichteffecten"); print ("1 - Looplicht cyclisch")
print ("2 - Looplicht heen en terug"); print ("3 - op- en neergaand")
print ("4 - allen knipperen gelijktijdig"); print ("5 - allen knipperen willekeurig")
print ("Strg+C beëindigt het programma")
```

Deze regels geven voor de gebruiker een handleiding op het beeldscherm, met welke cijfertoets welk patroon wordt weergegeven.



```
*Python Shell*
File Edit Shell Debug Options Windows Help
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Lichteffekte zur Auswahl
1 - Lauflicht zyklisch
2 - Lauflicht hin und zurück
3 - auf- und absteigend
4 - alle blinken gleichzeitig
5 - alle blinken zufällig
Strg+C beendet das Programm
Bitte Muster auswählen: 1
Bitte Muster auswählen: |
```

Afb. 5.3: Het programma op het beeldscherm.

Nadat de keuze is aangegeven, start de hoofdlus van het programma. Hiervoor gebruiken wij tevens hier een `while True:-gesloten lus`, die in `entry...except` aanwijzing is ingebed.

`e = raw_input ("Kies een voorbeeld a.u.b.: ")` Direct aan het begin van de lus wacht het programma op een gebruikersinvoer, die in de variabelen `e` wordt opgeslagen. De functie `raw_input()` neemt de invoer in normale tekst over zonder ze te beoordelen. In tegenstelling hierop, worden met `input()` ingevoerde wiskundige handelingen of namen van variabelen direct beoordeeld. In de meeste

gevallen is dus `raw_input()` de betere keuze, omdat men zich niet bezig hoeft te houden met veel eventualiteiten van mogelijke invoeren.

Het programma wacht, tot de gebruiker een letter invoert en op de `[Enter]`-toets drukt. Afhankelijk hiervan, welk getal de gebruiker heeft ingevoerd, moet nu een bepaald patroon met de LED's worden aangegeven. Om dit op te vragen, gebruiken wij een `if...elif...else`-constructie.

### Patroon 1

Was de invoer een 1, wordt het achter deze regel ingesprongen programmadeel uitgevoerd.

`if e == "1"`: Houd er rekening mee, dat insprongen in Python niet alleen het uiterlijke doel dienen. Net als bij lussen worden ook dergelijke vragen met een inspronging ingeleid.

#### Gelijk aan is niet onmiddellijk gelijk aan

Python gebruikt twee typen gelijkheidstekens. Het eenvoudige `=` is ervoor bedoeld, een bepaalde waarde aan een variabele toe te wijzen. Het dubbele gelijkheidsteken `==` wordt in vragen gebruikt en controleert, of twee waarden werkelijk gelijk zijn.

Indien de gebruiker dus een 1 heeft ingevoerd via het toetsenbord, start een lus, die een cyclisch looplicht creëert. Deze lussen zijn voor alle gebruikte LED-patronen principieel gelijk gebouwd.

`for i in range(w)`: De buitenste lus herhaalt het patroon zo vaak, als in de eerder gedefinieerde variabelen `w` is aangegeven. In deze lus bevindt zich nog één, die het betreffende patroon maakt. Dit is bij elk patroon verschillend.

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

Bij het eenvoudige cyclische looplicht loopt deze lus achtereenvolgend voor elke LED van de lijst een keer door. Hoeveel LED's dit zijn, is aan het begin van het programma in de variabelen `z` opgeslagen. De LED met het nummer van de actuele stand van de lusteller wordt ingeschakeld. Hierna wacht het programma de eerder in de variabelen opgeslagen `t` tijd en schakelt de LED vervolgens weer uit. Aansluitend begint de volgende lusdoorloop met de volgende LED. De buitenste lus herhaalt de totale binnenste lus vijf keer.

### Patroon 2

Als de gebruiker een 2 heeft ingevoerd, start een overeenkomstige lus. Hier worden de LED's echter niet alleen in één richting verder geteld, maar aan het einde van het looplicht weer in omgekeerde volgorde. Het licht loopt afwisselend heen en terug.

`elif e == "2"`: De overige vragen na de eerste keer gebruiken van de vraag `elif`, met de betekenis, dat ze alleen dan worden uitgevoerd, wanneer de vorige vraag als resultaat `False` heeft geretourneerd.

```

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)

```

Ook hier worden in elkaar geschakelde lussen gebruikt. Na de eerste binnenste lus, die overeenkomt met het eerder beschreven programmadeel, dus nadat de LED met nummer 3 brandt, start nog een lus voor het looplicht in tegengestelde richting. Lijstelementen zijn altijd met 0 beginnend genummerd. De vierde LED heeft dus het nummer 3.

Om een lus achterwaarts te laten lopen, gebruiken wij de uitgebreide syntax van `for...range()`. In plaats van slechts een eindwaarde aan te geven, kunnen ook drie parameters worden aangegeven: Startwaarde, stapgrootte en eindwaarde. In ons voorbeeld zijn dat:

Startwaarde	z-1	De variabele z bevat het aantal LED's. Omdat de nummering van de lijstelementen met 0 begint, heeft de laatste LED het nummer z-1.
Stapgrootte	-1	Bij een stapgrootte van -1 telt elke lusdoorloop een getal terug.
Eindwaarde	-1	De eindwaarde in een lus is altijd de eerste waarde, die niet wordt bereikt. In de eerste vooruit tellende lus begint de lusteller bij 0 en bereikt in ons voorbeeld de waarde 0, 1, 2, 3, om de LED's te adresseren. De 4 wordt bij vier keer een lusdoorloop niet bereikt. De achterwaarts tellende lus moet met 0 eindigen en zo de waarde -1 als eerste niet bereiken.

De beschreven tweede lus laat opeenvolgend de vier LED's in omgekeerde richting knipperen. Hierna start de buitenste lus van de totale cyclus opnieuw, die hier, omdat elke LED twee keer knippert, twee keer langer duurt dan in het eerst programmaonderdeel.

### Patroon 3

Als de gebruiker een 3 heeft ingevoerd, start een overeenkomstige lus. Hier worden de LED's ook in beide richtingen verder geteld, maar niet direct na het inschakelen weer uitgeschakeld.

```

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)

```

De eerste binnenste lus schakelt de LED's opeenvolgend met een vertraging in. Aan het einde van de lus, die kan worden herkend aan het uitspringen van de regel `time.sleep(2*t)`, wordt de dubbele vertragingstijd onderhouden. Gedurende die tijd branden alle LED's. Hierna begint een andere lus, die achterwaarts telt en de één na de andere LED weer uitschakelt. Ook hier wordt aan het einde, wanneer alle LED's uit zijn, de dubbele vertragingstijd onderhouden, voordat de buitenste lus de totale cyclus nog een keer start.

#### Patroon 4

Als de gebruiker een 4 heeft ingevoerd, start een ander knipperpatroon, waarbij alle LED's gelijktijdig knipperen en niet achtereenvolgend worden opgeroepen.

```
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)
```

Omdat niet meerdere GPIO-poorten met een enkele aanwijzing in één keer kunnen worden in- of uitgeschakeld, worden ook hier lussen gebruikt, echter zonder tijdsvertraging binnen de lus. De vier LED's worden direct na elkaar ingeschakeld. Voor het menselijke oog verschijnt het gelijktijdig. Aan het einde van de eerste binnenste lus onderhoudt het programma de dubbele vertragingstijd, voordat alle LED's weer worden uitgeschakeld.

Door verschillende aan- en uittijden kunnen verschillende effecten bij knipperlichten worden gecreëerd. Knipperen wordt eerder waargenomen, wanneer de aan-tijd langer duurt dan de uit-tijd. Zeer korte aan-tijden tijdens vergelijkbare lange uittijden creëren een knippereffect.

#### Patroon 5

Heeft de gebruiker een 5 ingevoerd, knipperen de LED's volledig willekeurig.

```
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)
```

Omdat hier geen geneste lussen worden gebruikt, laten we de lus vaker doorlopen. Door vermenigvuldiging van de variabelen `w` en `z` knippert elke LED gemiddeld zo vaak als in het eerste patroon.

De functie `random.randint()` schrijft een willekeurig getal in de variabele `j`. Dit willekeurig getal is groter of gelijk aan de eerste parameter en kleiner of gelijk aan de tweede parameter, kan dus in ons geval de waarden 0, 1, 2, 3 aannemen.

De willekeurig gekozen LED wordt in- en na de vertragingstijd weer uitgeschakeld. Hierna start de lus opnieuw en een nieuwe LED wordt willekeurig gekozen.

### Ongeldige invoer

Bij alle programma's, die invoer van gebruikers vereisen, moet foute invoer worden ondervangen. Wanneer de gebruiker iets invoert dat niet wordt verwacht, moet het programma hierop reageren.

```
else:  
    print ("Ongeldige invoer")
```

Als de gebruiker iets anders ingevoerd, wordt de onder `else` aangegeven aanwijzing uitgevoerd. Deze sectie van een vraag komt alleen dan overeen, wanneer geen van de andere vragen een waar resultaat heeft opgeleverd. In ons geval laat het programma een melding op het beeldscherm zien.

Net als in de voorgaande experimenten wordt het programma via een `KeyboardInterrupt` beëindigd, wanneer de gebruiker de toetscombinatie `Strg` + `C` indrukt. De laatste regel sluit de gebruikte GPIO-poorten en schakelt hiermee alle LED's uit.

## 6 LED via pulsduurmodulatie dimmen

LED's zijn typische onderdelen voor het uitvoeren van signalen in de digitale elektronica. Ze kunnen twee verschillende toestanden aannemen, aan en uit, 0 en 1 of `True` en `False`. Hetzelfde geldt voor de als digitale uitgangen gedefinieerde GPIO-poorten. Dus is het theoretisch niet mogelijk een LED te dimmen.

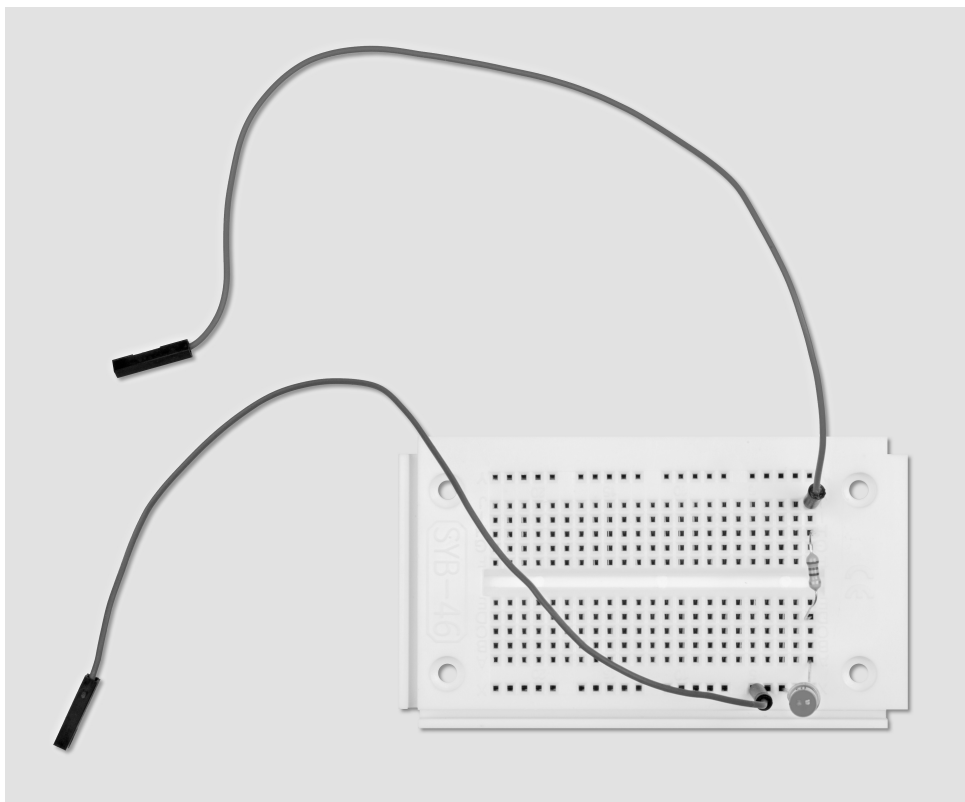
Met een trucje is het toch mogelijk, de lichtsterkte van een LED op een digitale GPIO-poort te regelen. Wanneer men een LED snel genoeg laat knipperen, neemt het menselijke oog dit niet meer waar als knipperen. De als pulsduurmodulatie gekenmerkte techniek creëert een pulserend signaal, dat kort achter elkaar aan- en uitschakelt. De spanning van het signaal blijft altijd gelijk, alleen de verhouding tussen level `False` (0 V) en level `True` (+3,3 V) wordt gewijzigd. De pulsverhouding geeft de lengte van de ingeschakelde toestand voor de totale duur van een schakelcyclus aan.



**Afb. 6.1:** Links: Pulsverhouding 50 % - rechts: Pulsverhouding 20 %.

Hoe kleiner de pulsverhouding, des te korter is de brandduur van de LED binnen een schakelcyclus. Hierdoor wordt de LED donkerder dan een permanent ingeschakelde LED

Sluit voor het volgende experiment een LED via een voorweerstand aan op GPIO-poort 18.



**Afb. 6.2:** Opbouw insteekbord met een LED

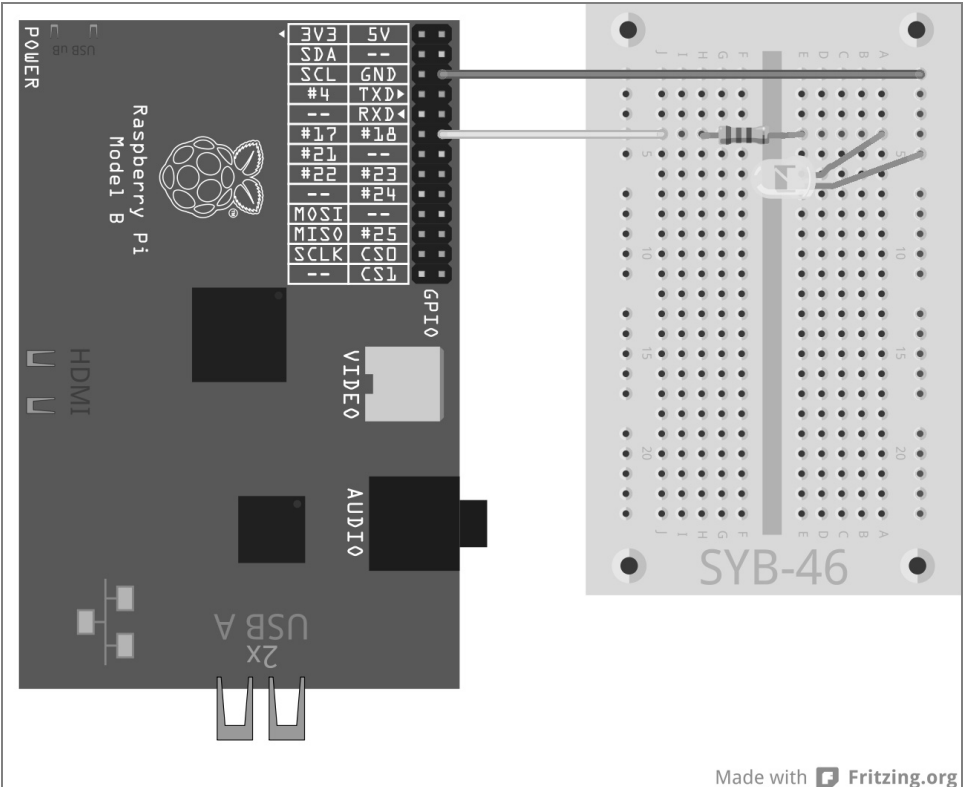
**Benodigde onderdelen:**

1x insteekprintplaat

1x LED geel

1x 220-ohm-weerstand

2x verbindingkabel



Made with Fritzing.org

**Afb. 6.3:** Een LED aan de GPIO-poort 18.

Het programma `leddimmen01.py` dimt de LED cyclisch lichter en donkerder en gebruikt hiervoor een eigen PWM-functionaliteit van de GPIO-bibliotheek. Het PWM-sigitaal wordt als eigen thread gegenereerd. Op deze manier kan een gedimde LED (bijna) als een normaal brandende LED in een programma worden toegepast.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM); LED = 18
GPIO.setup(LED, GPIO.OUT)
print ("Strg+C beëindigt het programma")
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 Zo werkt het

Een deel van dit programma zal u bekend voorkomen, enkele elementen echter helemaal niet, omdat wij op deze plaats een intermezzo in het objectgeoriënteerde programma maken. Aan het begin worden, zoals bekend, de bibliotheken geïmporteerd. Dit keer wordt slechts een enkele variabele, `LED`, voor de GPIO-port 18 vastgelegd, deze wordt als uitgang geïnitieerd.

```
print("Strg+C beëindigt het programma")
```

Omdat ook dit programma met een `try...except`-constructie loopt en door de gebruikers moet worden gestopt, wanneer betreffende informatie wordt weergegeven voor de gebruiker.

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

De functie `GPIO.PWM()` uit de GPIO-bibliotheek is beslissend voor uitvoer van PWM-signalen. Deze functie heeft twee parameters nodig, de GPIO-poort en de frequentie van het PWM-signaal. In ons geval wordt de GPIO-poort via de variabele `LED` vastgelegd, de frequentie is 50 Hertz (trillingen per seconde).

#### Waarom 50 Hertz de ideale frequentie is voor PWM

Het menselijke oog neemt lichtwisselingen sneller dan 20 Hertz niet meer waar. Omdat het wisselspanningsnet in Europa een frequentie van 50 Hertz gebruikt, knipperen veel verlichtingseenheden met deze frequentie, die niet meer door het oog wordt waargenomen. Wanneer een LED met meer dan 20 Hertz, maar minder dan 50 Hertz knippert, kunnen er interferenties ontstaan met andere lichtbronnen, waardoor het dimeffect niet meer gelijkmatig lijkt.

`GPIO.PWM()` creëert een zogenaamd object, dat in de variabele `p` wordt opgeslagen. Dergelijke objecten zijn veel meer dan alleen eenvoudige variabelen. Objecten kunnen verschillende eigenschappen bezitten en door zogenaamde methoden worden beïnvloed. Methoden worden, door een punt gescheiden, direct achter de objectnaam aangegeven.

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

De methode `start()` start het genereren van het PWM-signaal. Hiervoor moet nog een pulsverhouding worden aangegeven. In ons geval is de pulsverhouding 0, de LED is dus altijd uitgeschakeld. Nu start de gesloten lus, waarin direct achtereenvolgend twee lussen zijn ingebouwd, die afwisselend de LED lichter en donkerder laten worden.

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

De lus telt in stappen van 2 van 0 tot en met 100. Als einde van een `for`-lus wordt altijd de waarde aangegeven, die net niet wordt bereikt, in ons geval 101.

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

In elke lusdoorloop zet de methode `ChangeDutyCycle()` de pulsverhouding van het PWM-object op de waarde van de lusteller, dus elke keer 2% hoger, tot het tijdens de laatste doorloop op 100% staat en de LED met volle lichtsterkte brandt.

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

In elke lusdoorloop wordt 0,1 seconde gewacht, voordat de volgende doorloop de pulsverhouding weer met 2% verhoogd.

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



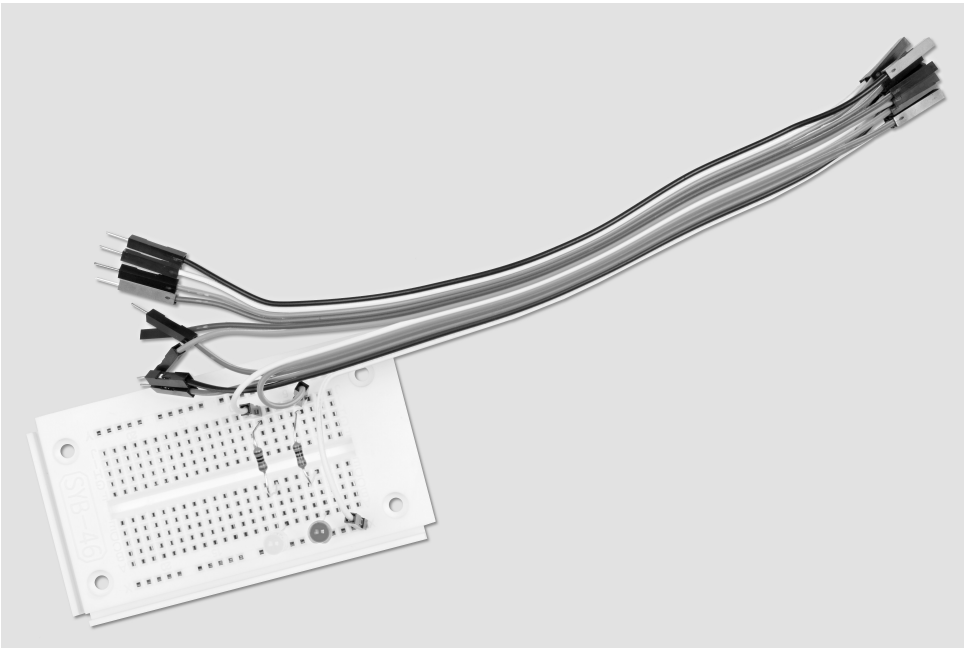
Nadat de LED de volledige lichtsterkte heeft bereikt, regelt een tweede lus haar volgens hetzelfde schema weer omlaag. Deze lus telt van 100 in stappen van -2 omlaag. Deze cyclus herhaalt zich tot een gebruiker de toetscombinatie `Strg`+`C` indrukt.

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

De `KeyboardInterrupt` activeert nu aanvullende de methode `stop()` van het PWM-object. Deze methode beëindigt de aanmaak van een PWM-sigitaal. Hierna worden, zoals uit het laatste programma bekend, de GPIO-poorten teruggezet.

### 6.1.2 Twee LED's onafhankelijk van elkaar dimmen

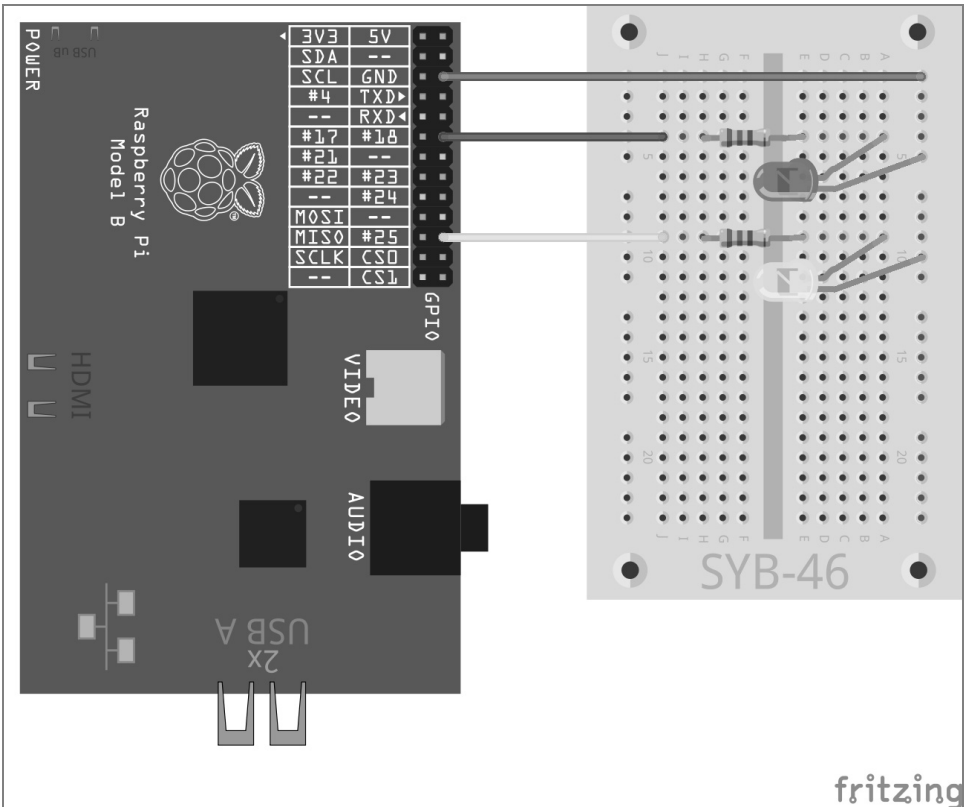
Omdat voor het programmeren van het PWM-sigitaal geen programmatijd in het Python-script nodig is, kunnen ook meerdere LED's onafhankelijk van elkaar worden gedimd, zoals het volgende experiment laat zien. Sluit hiervoor nog een LED via een voorweerstand aan op GPIO-poort 25.



**Afb. 6.4:** Opbouw insteekbord, omtwee LED's te dimmen.

Benodigde onderdelen:

- 1x insteekprintplaat
- 1x LED geel
- 1x LED rood
- 2x 220-ohm-weerstand
- 3x verbindingskabel



Afb. 6.5: Een tweede LED aan de GPIO-poort 25.

Het programma `leddimmen02.py` dimt een LED cyclus lichter en donkerder, terwijl de andere LED weliswaar samen met de eerste LED lichter, maar in de andere cyclus niet donkerder wordt, maar weer vanaf 0 lichter wordt en hierbij snel flinkt.

```
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 beëindigt het programma")

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 Zo werkt het

De basisstructuur van het programma komt overeen met het vorige experiment met een paar kleine uitbreidingen.

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

In plaats van een variabele voor de GPIO-poort wordt nu een lijst met twee variabelen gedefinieerd, en hiermee worden twee GPIO-poorten 18 und 25, als uitgangen voor de LED's geïnitieerd.

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

Aansluitend worden ook de twee objecten `p` en `q` aangemaakt, die de PWM-signalen voor de beide LED's met steeds 50 Hertz creëren.

```

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

```

In de eerste lus worden de pulsverhoudingen van de beide PWM-objecten gelijktijdig stap-voor-stap verhoogd. De beide LED's gedragen zich in deze fase gelijk.

`q.ChangeFrequency(10)` Aan het einde van deze lus, wanneer de beide LED's de volledige lichtsterkte hebben bereikt, wordt de frequentie van het PWM-signaal van de tweede LED via de methode `ChangeFrequency()` naar 10 Hertz verlaagd. Deze frequentie wordt door het oog nog als knipperen waargenomen.

```

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

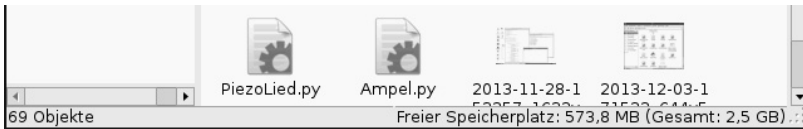
```

Nu start de tweede lus, omwille van het overzicht dit keer ook met oplopende telling. Voor de eerste LED uit het PWM-object `p`, die in deze cyclus stap-voor-stap moet worden gedimd, worden de betreffende waarden voor de pulsverhouding in elke doorloop berekend. Bij de tweede LED uit het PWM-object `q` wordt de pulsverhouding eenvoudig weer bijgeteld. Het knippereffect ontstaat door de gewijzigde frequentie.

`q.ChangeFrequency(50)` Aan het einde van de tweede lus wordt de frequentie van deze LED weer naar 50 Hertz teruggezet, opdat ze in de volgende cyclus weer precies als de eerste LED langzaam lichter wordt.

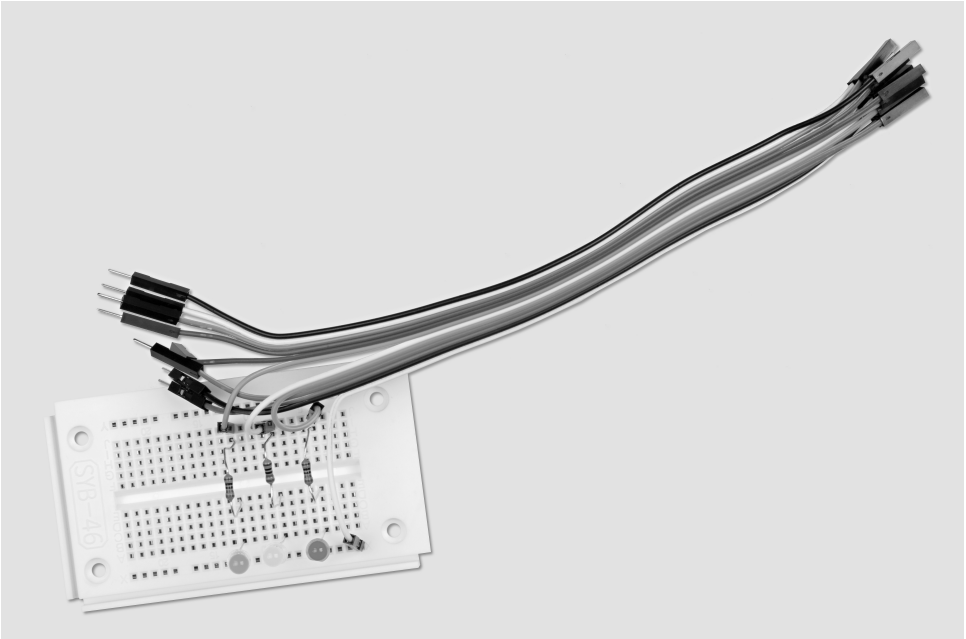
## 7 Ruimteweergave van geheugenkaart met LED's

Geheugen zijn net als harde schijven altijd snel vol. Hier wil men graag een eenvoudige optische ruimteweergave, om altijd in één oogopslag te kunnen zien, wanneer de geheugenruimte bijna is gebruikt. Met drie LED's kan dit op de Raspberry Pi zeer eenvoudig worden gerealiseerd. Hiervoor worden functies van het besturingssysteem gebruikt, die via Python worden opgeroepen.



**Afb. 7.1:** Natuurlijk kan de vrije geheugenkaartruimte ook direct in de verkener op de Raspberry Pi worden weergegeven.

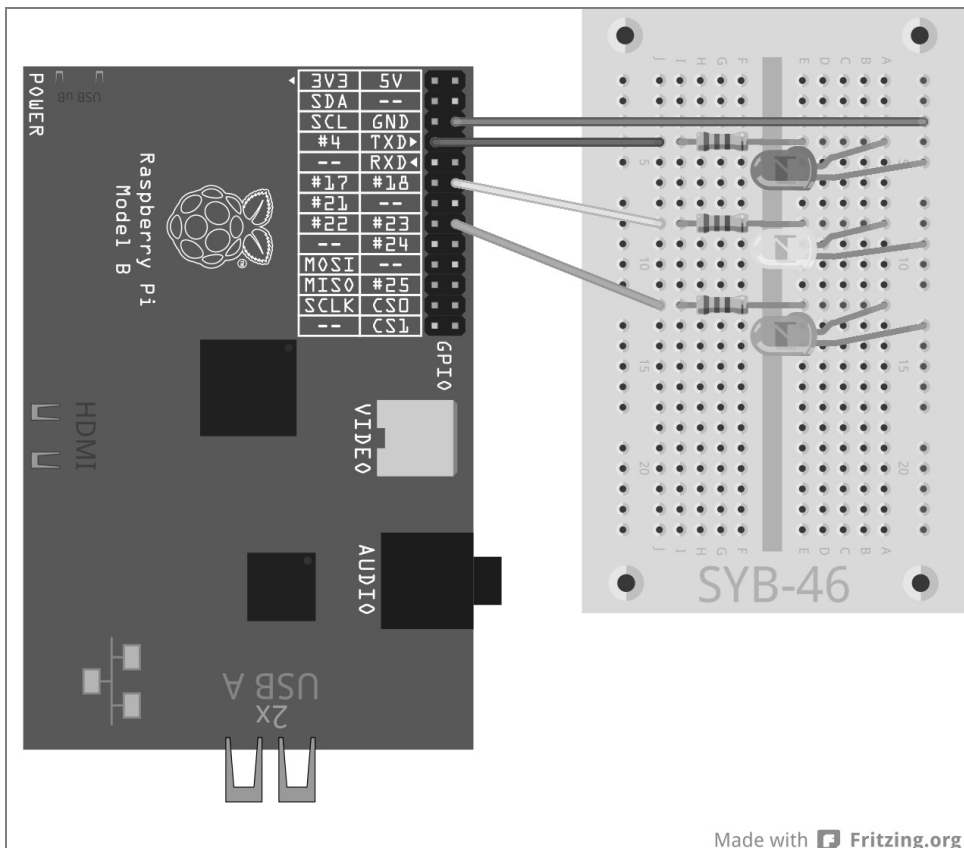
Voor de weergave van de vrije geheugenkaartruimte gebruiken wij de drie LED's uit de verkeerslichtschakeling, die in verschillende kleurcombinaties branden.



**Afb. 7.2:** Opbouw insteekbord voor de ruimteweergave van de geheugenkaart.

**Benodigde onderdelen:**

- 1x insteekprintplaat
- 1x LED rood
- 1x LED geel
- 1x LED groen
- 3x 220-ohm-weerstand
- 4x verbindingkabel



Made with  Fritzing.org

Afb. 7.3: De drie LED's geven de vrije geheugenruimte op de geheugenkaart weer.

Het programma `speicheranzeige.py` levert afhankelijk van de vrije geheugenruimte op de geheugenkaart verschillende LED-indicaties:

Vrije geheugenruimte	LED-indicator
< 1 MB	Rood
1 MB tot 10 MB	rood-geel
10 MB tot 100 MB	geel
100 MB tot 500 MB	geel-groen
> 500 MB	groen

Tab. 7.1: Zo wordt de vrije ruimte van de geheugenkaart weergegeven.

```

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 beëindigt het programma")

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()

```

Als u het programma laat lopen, geven de LED's voortdurend de vrije geheugenruimte op de geheugenkaart weer. Probeer dit uit, door grote bestanden via het netwerk op de geheugenkaart te kopiëren en weer te verwijderen. De weergave wordt automatisch bijgewerkt.

### 7.1.1 Zo werkt het

Het programma gebruikt de Python-module `os` voor het berekenen van de vrije geheugenruimte, dat de fundamentele besturingssysteemfuncties ter beschikking stelt.

`import os` De module `os` moet, net als andere modules, aan het begin van het programma worden geïmporteerd.

`g1 = 1; g2 = 10; g3 = 100; g4 = 500` Deze regels definiëren de grenzen van de gebieden voor vrije geheugenruimte, vanwaar de indicatie moet omschakelen. Vanwege de eenvoud gebruikt het programma megabyte en niet byte, omdat men zich deze getallen beter kan inbeelden. De grenzen kunnen te allen tijde anders worden vastgelegd, de vier waarden moeten alleen in oplopende grootte worden toegekend.

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

Een lijst definieert de GPIO-nummer van de drie LED's. Vervolgens initialiseert een lus de drie GPIO-poorten als uitgangen en zet alle LED's op uitgeschakeld.

Ook in dit experiment gebruiken wij een `try...except`-constructie en een gesloten lus, om het programma steeds weer automatisch te laten lopen, tot de gebruiker het met `Strg` + `C` afbreekt. Hierna volgen de eigenlijke interessante functies, die toegang hebben op het besturingssysteem en de vrije geheugenruimte oproepen.

`s = os.statvfs('/')` De statistische module `os.statvfs()` uit de `os`-bibliotheek levert diverse statische informatie voor het bestandssysteem, die hier binnen de gesloten lus bij elke lusdoorloop opnieuw als object in de variabele `s` worden geschreven.

`f = s.f_bsize * s.f_bavail / 1048576` Nu levert de methode `s.f_bsize` de grootte van een geheugenblok in byte. `s.f_bavail` geeft het aantal vrije blokken weer. Het product uit beide waarden geeft derhalve het aantal vrije bytes weer, die hier door 1.048.576 wordt gedeeld, om het aantal vrije megabytes verkrijgen. Het resultaat wordt in de variabele `f` opgeslagen.

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

Indien de vrije geheugenruimte kleiner is dan de eerste grenswaarde (1 MB), wordt de tekenvolgorde `x`, die het patroon van de ingeschakelde LED aangeeft, op "100" gezet. De eerste, rode LED moet branden. Het patroon is een eenvoudige tekenketting uit de cijfers 0 en 1.

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

Met behulp van `elif`-vragen, worden de andere grenswaarden opgeroepen en het LED-patroon overeenkomstig ingesteld, wanneer de eerste vraag niet overeenkomt, dus meer dan 1 MB vrije geheugenruimte beschikbaar is.

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

Indien geen van de vragen overeenkomt, dus meer vrije geheugenruimte beschikbaar is, dan de hoogste grenswaarde aangeeft, wordt het LED-patroon op "001" gezet. De laatste, groene LED moet branden.

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



Een lus legt de GPIO-uitvoerwaarden voor de drie LED's vast. Opeenvolgend krijgen alle LED's de getalwaarde van het betreffende cijfer uit de tekenvolgorde 0 of 1 toegekend. De waarden 0 en 1 kunnen evenals `False` en `True` worden gebruikt, om GPIO-uitgangen uit- of in te schakelen. De functie `int()` berekent uit een teken de getalwaarde hiervan. Het teken wordt via de lusteller `i` uit een bepaalde positie van de patroontekenketting gelezen.

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

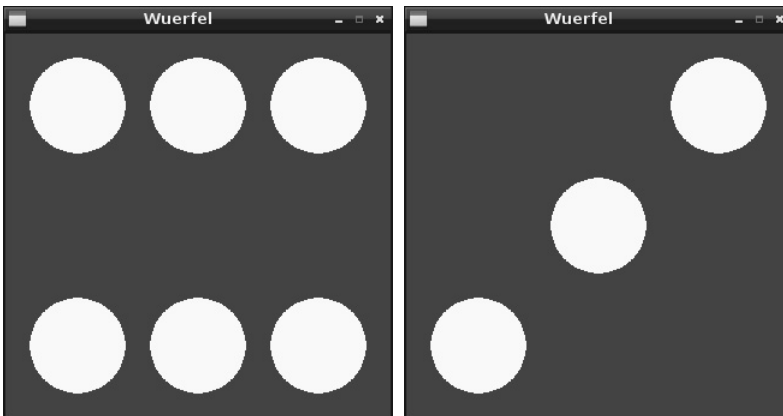
Het programma wacht 1 seconden tot aan de volgende lusdoorloop. Om vermogen te besparen, kunnen ook langere wachttijden worden vastgelegd, tot de berekening van de vrije geheugenruimte moet worden herhaald.

Op deze plaats begint de `while...True`-lus opnieuw. Indien de gebruiker in de tussentijd de toetscombinatie `[Strg] + [C]` hebben ingedrukt, wordt een `KeyboardInterrupt` geactiveerd en de lus wordt verlaten. Vervolgens worden de GPIO-poorten gesloten en derhalve de LED's uitgeschakeld.

## 8 Grafische dobbelsteen

Een stoer spel heeft grafiek nodig en niet alleen een tekstuitvoer zoals in het tijdperk van de allereerste DOS-computers. De bibliotheek PyGame levert vooraf gedefinieerde functies en objecten voor de grafische weergave en spelprogrammering. Hierdoor hoeft men alles niet meer vanaf de basis opnieuw uit te vinden.

Voor veel spellen heeft men een dobbelsteen nodig, maar vaak is er geen binnen handbereik. Het volgende programmavoorbeeld laat zien, hoe gemakkelijk het is, de Raspberry Pi met behulp van Python en PyGame als dobbelsteen te gebruiken:



Afb. 8.1: De Raspberry Pi als dobbelsteen.

De dobbelsteen moet mogelijk eenvoudig en met slechts één toets kunnen worden bediend, en het willekeurig gedobbelde resultaat moet grafisch als een »echte« dobbelsteen worden weergegeven. Het onderstaande programma `wuerfel.py` simuleert een dergelijke dobbelsteen op het beeldscherm.

```

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

VELD = pygame.display.set_mode((320, 320))
pygame.display.set_caption("dobbelsteen")

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

print "Druk een willekeurige toets in, om de dobbelsteen te gooien, [Esc]
beëindigt het spel"

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

```

## Loopt zonder sudo

Omdat dit programma geen GPIO-poorten nodig heeft, loopt het ook zonder superuser-rechten. U kunt de Python-IDLE eenvoudig via het desktop-pictogram *IDLE* starten.

### 8.1.1 Zo werkt het

Dit programma toont een veelvoud aan nieuwe functies, in het bijzonder voor de grafische weergave met de PyGame-bibliotheek, die natuurlijk niet alleen voor spellen, maar ook voor alle andere grafische weergaves op het beeldscherm kan worden gebruikt.

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

Deze drie programmaregels staan aan het begin van bijna elk programma, dat PyGame gebruikt. Naast de al genoemde module `random` voor het genereren van willekeurige getallen wordt de module `pygame` zelf alsmede de module `sys` geladen, omdat het belangrijke, voor PyGame noodzakelijke systeemfuncties bevat, zoals bijv. het openen en sluiten van schermen. Alle functies uit de PyGame-bibliotheek worden geïmporteerd en hierna wordt de eigenlijke PyGame-module geïnitieerd.

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

Deze belangrijke functie in elk programma, dat een grafische uitvoer gebruikt, definieert een tekenvlak, een zogenaamde surface, die in ons voorbeeld 320 x 320 pixels groot is en de naam `FELD` krijgt. Let op de schrijfwijze tussen dubbele haakjes, die principieel gebruikt wordt voor grafische beeldschermcoördinaten. Een dergelijke surface wordt in een nieuw scherm op het beeldscherm weergegeven.

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

Deze regel voert de beeldschermnaam in.

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

Deze regels definiëren de beide gebruikte kleuren blauw en wit. Men kan ook elke keer in het programma de kleurwaarde direct aangeven, wat echter niet bijdraagt aan de overzichtelijkheid.

## Weergave van kleuren op het beeldscherm

Kleuren worden in Python, evenals in de meeste andere programmeertalen, door drie getallen tussen 0 en 255 gedefinieerd, die de drie kleuraandelen rood, groen en blauw vastleggen. Beeldschermen gebruiken een additieve kleurmenging, waarbij alle drie kleuraandelen bij volle verzadiging in wit resulteren.

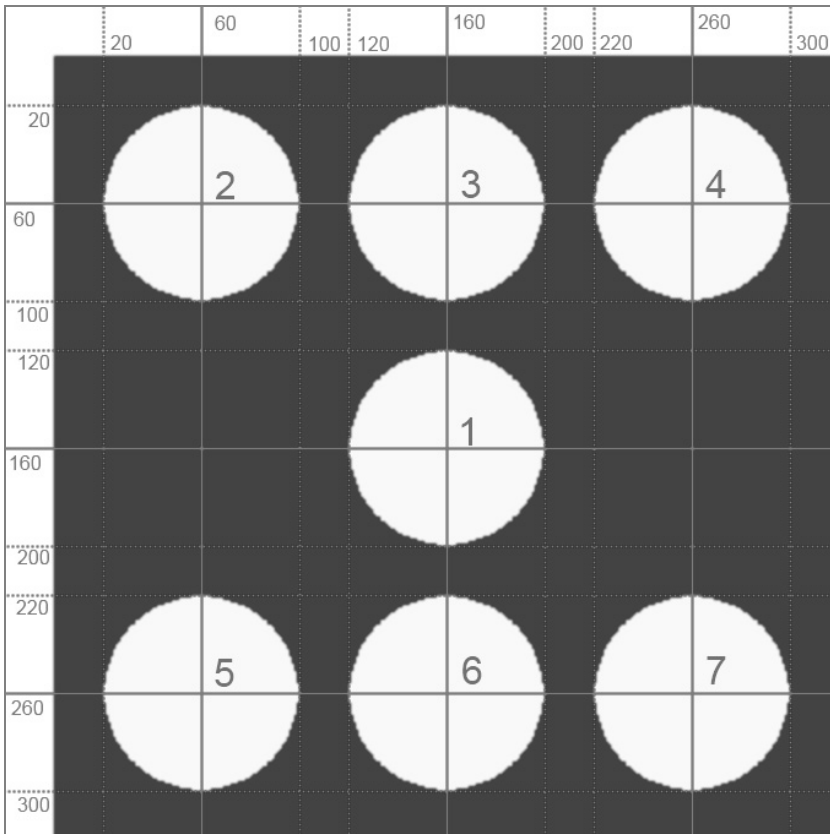
```
P1 = ((160, 160)); P2 = ((60, 60)); P3 = ((160, 60)); P4 = ((260, 60)); P5 = ((60, 260)); P6 = ((160, 260)); P7 = ((260, 260))
```

Deze regels bepalen het middelpunt van de dobbelsteenogen. Op het 320 x 320 pixels grote tekenveld liggen de drie assen van de dobbelsteenogen steeds op de coördinaten 60, 160 en 260.

### Het coördinatensysteem voor computergrafiek

Ieder punt in een scherm resp. op een surface-object wordt door een  $x$ - en een  $y$ -coördinaat gekenmerkt. Het nulpunt van het coördinatensysteem is niet, zoals men het op school heeft geleerd, links onder, maar linksboven. Net als dat men een tekst van linksboven naar rechtsonder leest, loopt de  $x$ -as van links naar rechts, de  $y$ -as van boven naar onderen.

De zeven punten  $P_1$  tot en met  $P_7$  kenmerken de in de grafiek aangegeven middelpunten van de dobbelsteenogen. Elk dobbelsteen oog heeft een radius van 40 pixels. Bij een asafstand van 80 pixels, blijven derhalve 20 pixels tussen de dobbelsteenogen en 20 pixels tot de schermranden.



Afb. 8.2: De dobbelsteenogen en hun coördinaten

Op deze plaats wordt met de andere variabelen ook nog een variabele namens `mainloop` op `True` gezet, die later nodig is voor de hoofdloop van het spel.

`mainloop = True` Hiermee is de basis gemaakt, en het eigenlijke spel kan beginnen.

```
print "Druk op een willekeurige toets, om te dobbelen, [Esc]  
beëindigt het spel"
```

Deze regel legt in het kort uit aan de gebruiker, wat er moet worden gedaan. Bij elke keer indrukken van een willekeurige toets op het toetsenbord, wordt opnieuw gedobbeld. `print` schrijft altijd in het Python-shell-scherm, niet in het nieuwe grafische scherm.

`while mainloop`: Nu begint de hoofdlus van het spel. In veel spellen wordt een gesloten lus gebruikt, die zich steeds weer herhaalt en voortdurend enigerlei gebruikersactiviteit oproept. Ergens in de lus moet een onderbrekingsvoorwaarde zijn gedefinieerd, die er voor zorgt, dat het spel kan worden beëindigd.

Hiervoor wordt de variabele `mainloop` gebruikt, die alleen de beide boole-waarden `True` en `False` (`True` en `Falsch`, aan en uit) aanneemt. Aan het begin staat ze op `True` en wordt bij iedere lusdoorloop opgeroepen. Heeft ze tijdens de lus de waarde `False` aangenomen, wordt de lus voor de volgende doorloop beëindigd.

`for event in pygame.event.get()`: Deze regel leest de laatste gebruikersactiviteit en slaat haar op als `event`. In het spel zijn er slechts twee manieren met spelrelevante gebruikersactiviteiten: De gebruiker drukt op een toets en dobbelt hiermee, of de gebruiker wil het spel beëindigen.

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

Er zijn twee mogelijkheden om het spel te beëindigen: Men kan op het x-symbool in de rechter bovenhoek van het scherm klikken, of op de toets `[Esc]` drukken. Wanneer men op het x-symbool klikt, is het de vanuit het besturingssysteem geleverde `event.type == QUIT`. Wanneer men op een toets drukt en weer loslaat, is het `event.type == KEYUP`. Daarnaast wordt in dit geval de ingedrukte toets in `event.key` opgeslagen.

De beschreven `if`-vraag controleert, of de gebruiker het scherm wil sluiten of (`or`) een toets heeft ingedrukt en losgelaten en (`and`) dit de toets is met het interne kenmerk `K_ESCAPE`. Als dit het geval is, wordt de variabele `mainloop` op `False` gezet, waardoor de hoofdlus van het spel voor de volgende doorloop wordt beëindigd.

`if event.type == KEYDOWN`: Het tweede type gebruikersactiviteit, dat tijdens het spel steeds weer en niet slechts één keer voorkomt is, dat de gebruiker een toets indrukt. Hierbij is het onbelangrijk welke, met uitzondering van de `[Esc]`-toets, welke dit is. En wanneer een toets is ingedrukt (`KEYDOWN`), wordt een belangrijk programmaonderdeel opgestart, die het dubbelresultaat aanmaakt en ook weergeeft.

`VELD.fill(BLAUW)` Als eerste wordt het als `FELD` gekenmerkte `surface-object`, het eigenlijke programma-scherm, met de in het begin als `BLAU` gedefinieerde kleur gevuld, om het voorgaande dubbelresultaat over te verven.

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

Nu genereert de willekeursfunctie `random` een willekeurig getal tussen 1 en 6 en slaat deze op in de variabele `ZAHL`.

`print ZAHL` Deze regel schrijft alleen ter controle het dubbelresultaat in het Python-shell-scherf. Deze regel kan ook worden weggelaten, wanneer u op de op tekstgebaseerde uitvoer wilt afzien.

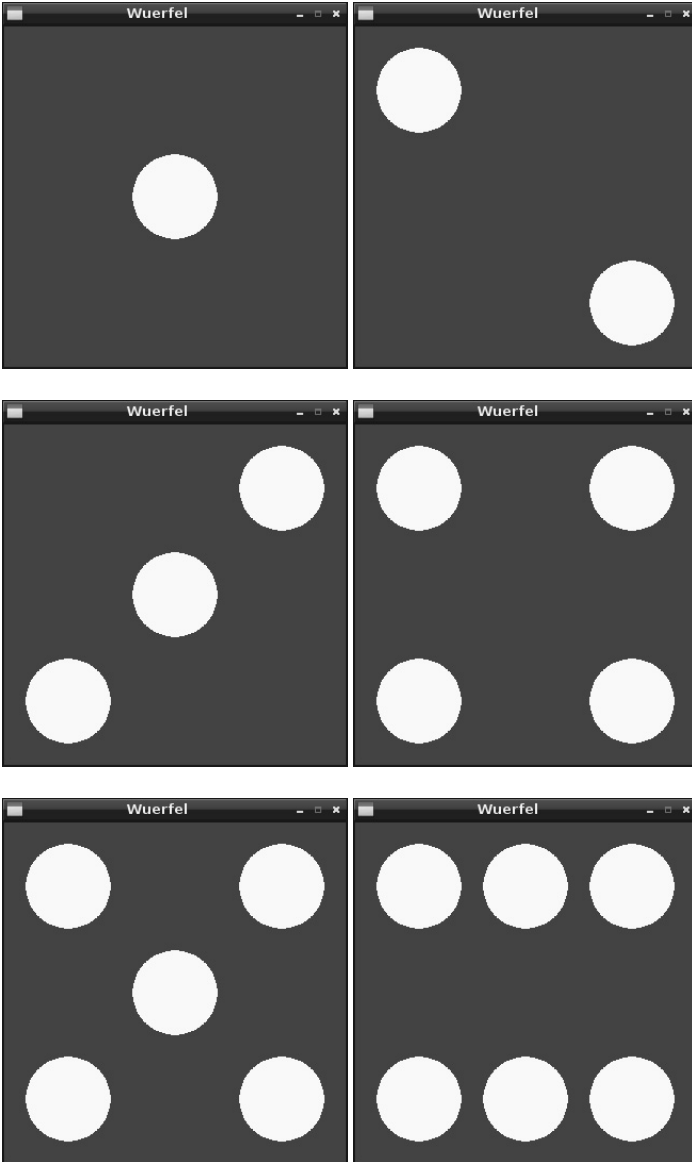
```
if ZAHL == 1:  
    pygame.draw.circle(FELD, WEISS, P1, 40)
```

Nu volgen er zes vragen, allen volgens hetzelfde schema. Wanneer het willekeurig gedubbelde getal een bepaalde waarde heeft, worden dienovereenkomstig één tot en met dubbelsteenogen getekend. De hiervoor gebruikte functie `pygame.draw.circle()` heeft vier of vijf parameters nodig:

- *Surface* geeft het tekenvlak weer, waarop wordt getekend, in het voorbeeld het `FELD`.
- *Kleur* geeft de kleur van de cirkel aan, in het voorbeeld de voorafgaand gedefinieerde kleur `WEISS`.
- *Middelpunt* geeft het middelpunt van de cirkel aan.
- *Radius* geeft de radius van de cirkel aan.
- *Dikte* geeft de lijndikte van de cirkellijn aan. Wordt deze parameter weggelaten of op 0 gezet, wordt de cirkel gevuld.

Is aan een van de *if*-voorwaarden voldaan, zijn de dubbelsteenogen vervolgens alleen op een virtueel tekenvlak opgeslagen.

`pygame.display.update()` Pas deze regel aan het luseinde actualiseert de grafiek op het beeldscherm. Nu zijn de dubbelsteenogen werkelijk zichtbaar.



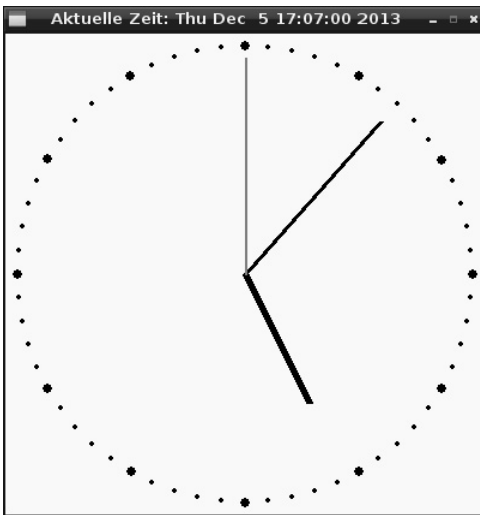
Afb. 8.3: De zes mogelijke dobbelresultaten.

De lus start direct vanaf het begin en wacht weer op het indrukken van een toets door de gebruiker. Wanneer tijdens de lus `mainloop` op `False` is gezet, omdat de gebruiker het spel wil beëindigen, wordt de lus niet nog een keer doorlopen, maar in plaats daarvan wordt de volgende regel uitgevoerd.

`pygame.quit()` Deze regel beëindigt de PyGame-module, waardoor tevens het grafische scherm sluit en daarna het volledige programma.

## 9 Analoge klok op het beeldscherm

De digitale tijdweergave, zoals wij het nu gewend zijn op computers, is pas in de jaren 70 in de mode gekomen. Voordien heeft men gedurende honderden jaren de tijd analoog met wijzers weergegeven op een wijzerplaat. De digitale klokboom is in de laatste jaren weer iets teruggekomen, omdat men heeft herkend, dat analoge klokken sneller en bij slechte weersomstandigheden of op grote afstanden, zoals op treinstations, ook beter kunnen worden afgelezen. Het menselijke oog registreert een grafiek sneller dan cijfers of letters. Het beeld van een analoge klok wordt door het kortetermijngeheugen onthouden, zodat men het, ook wanneer men het slechts onvolledig of wazig heeft gezien, nog juist kan interpreteren. Ziet men daarentegen een digitale klok slechts onduidelijk, kan men hieruit geen betrouwbare gevolgtrekkingen met betrekking tot de weergegeven tijd maken.



Afb. 9.1: Analoge klok met PyGame geprogrammeerd.

Dit programma moet niet alleen weergeven, hoe men een klok programmeert, maar ook fundamentele principes voor de weergave van analoge aanduidingen uitleggen, hoe ze niet alleen voor klokken maar ook voor de weergave van veel verschillende meetwaarden of statische gegeven kunnen worden gebruikt.

Om het middelpunt van de ronde wijzerplaat lopen drie wijzers, die het uur, de minuut en de seconde aangeven. Boven in het scherm loopt bovendien nog een digitale tijdsaanduiding mee.



Het programma `uhr01.py` geeft de afgebeelde analoge klok weer op het beeldscherm:

```
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("Actuele tijd: " +
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 Zo werkt het

Dit programma laat meer functies van de PyGame-bibliotheek zien en van de `time`-bibliotheek, alsmede eenvoudige trigonometrische hoekfuncties, die voor de weergave van analoge aanduidingen worden gebruikt.

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

Aan het begin wordt net als in het laatste programma de PyGame-bibliotheek geïnitieerd. Daarnaast worden de `time`-bibliotheek voor de tijdbepaling alsmede drie functies uit de zeer uitgebreide `math`-bibliotheek geïmporteerd.

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

De drie in de grafiek gebruikte kleuren worden in drie variabelen opgeslagen.

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

Een 400 x 400 pixel groot scherm wordt geopend en volledig met wit ingevuld.

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

Drie variabelen leggen de coördinaten van het middelpunt vast, waarop de betrokken verdere grafische elementen, de wijzerplaat en de wijzers worden uitgelijnd. De variabelen *MX* en *MY* bevatten de x- en y-coördinaten van het middelpunt, de variabele *MP* het middelpunt als punt, zoals hij voor grafische functies wordt gebruikt.

Als volgende volgt de definitie van een belangrijke functie, die aan de hand van de afstand tot het middelpunt en een hoek, punten in het coördinatiesysteem berekend. Deze functie wordt voor de weergave van zowel de wijzerplaat alsook de wijzers meerdere keren in het programma opgeroepen.

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

De functie gebruikt twee parameters: *A* is de afstand van het gewenste punt vanaf het middelpunt, *W* de hoek gerelateerd aan het middelpunt. Om de weergave in het geval van de klok te vergemakkelijken, nemen we de hoek met de wijzers van de klok mee op de verticale 12-uur-richting aan. De hoek wordt ook niet in graden maar in minuten, 1/60 van een volledige cirkel, aan de functie overgegeven. Dergelijke veronderstellingen besparen veel tussenberekeningen.

Python rekt zoals de meeste programmeertalen hoekeenheden in boogmaat en niet in graad. De functie `radian()` uit de `math`-bibliotheek, rekt graad om in boogmaat. Hiervoor wordt de in het programma gebruikte hoekinformatie in minuten met 6 vermenigvuldigd, om de graad te berekenen, en aansluitend wordt 90 graden afgetrokken, opdat de 0-richting verticaal naar boven wijst, zoals de minuut 0 van elk uur. Deze hoek omgerekend in boogmaat wordt voor verdere berekeningen binnen de functie in de variabele *w1* opgeslagen.

De weergave van een analoge klok is gebaseerd op de hoekfuncties sinus en cosinus. Hiermee worden uit de hoek van een punt in een boogmaat tegenover het middelpunt (*w1*) de coördinaten in een rechthoekig coördinatensysteem (*x1* en *y1*) bepaald. De coördinaten van het middelpunt worden uit de variabelen *MX* en *MY* overgenomen, die buiten de functie zijn gedefinieerd en algemeen gelden. De afstand van het punt vanaf het middelpunt wordt via de parameter *A* aan de functie overgedragen. De functie `int()` bepaalt uit het resultaat de integerwaarde (gehele getal), omdat pixelcoördinaten alleen als integer kunnen worden aangegeven.

De retourwaarde van de functie is een geometrisch punt met de berekende coördinaten *x1* en *y1*, die evenals alle punten tussen dubbele haakjes wordt gezet.

Na het definiëren van deze functie wordt de wijzerplaat getekend.

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

Een lus tekent opeenvolgend de 60 minutenpunten op een cirkel. Alle punten worden met de functie `punt()` bepaald. Ze hebben de gelijke afstand vanaf het middelpunt, dat met 190 pixels in de vier kwadranten nog exact 10 pixels van de schermrand is verwijderd. De punten hebben een radius van 2 pixels.

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

Een tweede lus tekent 12 grotere cirkels, die de uren op de wijzerplaat markeren. Deze hebben een radius van 4 pixels, worden eenvoudig via de aanwezige cirkels getekend en bedekken deze volledig. De uursymbolen volgen op een hoekafstand van vijf minuteneenheden op elkaar, dat door de vermenigvuldiging met 5 in de hoekinformatie, die aan de functie wordt overgedragen, wordt bereikt.

`mainloop = True; s1 = 0` Voordat de hoofdloop het programma start, worden nog twee hulpvariabelen gedefinieerd, die in het volgende verloop nodig zijn. `mainloop` geeft evenals in het laatste programmavoorbeeld weer, of de lus verder moet lopen, of dat de gebruiker het programma wil beëindigen. `s1` slaat de als laatste weergegeven seconde op.

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

Nu start de hoofdloop het programma, dat in iedere doorloop, onafhankelijk van hoe lang hij duurt, de actuele tijd in het object `zeit` schrijft. Hiervoor wordt de functie `time.localtime()` uit de `time`-bibliotheek gebruikt. Het resultaat is een gegevensstructuur, die uit verschillende individuele waarden bestaat.

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

De drie voor de wijzers relevante waarden, seconden, minuten en uren, worden uit de structuur in drie variabelen `s`, `m` en `h` geschreven.

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

Analoge klokken geven slechts twaalf uren weer. De functie `time.localtime()` levert alle tijdsaanduidingen in 24-uur-formaat. Bij tijdsaanduidingen na 12 uur 's middags worden dus eenvoudig 12 uren afgetrokken.

### Tijdweergave bij analoge klokken

Afhankelijk van het gebruikte mechaniek zijn er twee verschillende weergaven bij analoge klokken. Bij werkelijk analoog lopende klokken voert de minutenwijzer een gelijkvormige cirkelbeweging uit, bij digitaal aangestuurde klokken, zoals bijvoorbeeld stationsklokken, springt hij op de volle minuut een volledige minuut vooruit. De laatste werkwijze heeft als voordeel, dat de tijd beter in een oogopslag op de minuut nauwkeurig kan worden gelezen. Breukdelen van minuten zijn in het dagelijkse leven doorgaans niet belangrijk. Wij gebruiken voor ons urenprogramma eveneens dezelfde werkwijze. De uurwijzer moet echter een gelijkvormige cirkelbeweging uitvoeren. Het zou hier zeer vreemd en onoverzichtelijk zijn, wanneer hij elk hele uur een uur vooruit springt.

$hm = (h + m / 60.0) * 5$  De variabele `hm` slaat de hoek van de urenwijzers op in minuteneenheden, zoals ze in het hele programma worden gebruikt. Hiervoor wordt voor het actuele uur  $1/60$  van de minutenwaarde

opgeteld. In elke minuut beweegt de urenwijzer  $1/60$  uur verder. De berekende waarde wordt met 5 vermenigvuldigd, omdat de urenwijzer in een uur vijf minuteneenheden op de wijzerplaat vooruitgaat.

`if s1 <> s`: De duur van een lusdoorloop in een programma is niet bekend. Voor de analoge klok betekent dit, dat de grafiek niet bij elke lusdoorloop moet worden bijgewerkt, echter alleen, wanneer de actuele seconde afwijkt van degene die als laatste is getekend. Hierover wordt later in het programma de getekende seconde in de variabele `s1` opgeslagen, de actuele seconde staat altijd in de variabele `s`.

Wanneer de seconde zich ten opzichte van de als laatste getekende heeft gewijzigd, wordt met de volgende aanwijzingen de grafiek van de klok bijgewerkt. Heeft zij zich niet gewijzigd, is het niet nodig de grafiek bij te werken, en de lus start opnieuw met een verdere vraag van de actuele systeemtijd.

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

Als eerste wordt een wit cirkelvlak getekend, die de uurwijzer volledig bedekt. De radius is met 182 pixels iets groter dan de langste wijzer, opdat hiervan geen resten meer blijven staan. Het tekenen van een volledig vlakbedekkende cirkel, is duidelijk gemakkelijker, dan de als laatst getekende wijzer weer op de pixel nauwkeurig over te verven.

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

Deze regel tekent de urenwijzer als lijn met een breedte van 6 pixels, vanuit het middelpunt 120 pixels lang in een hoek, die door de variabele `hm` wordt aangegeven. De functie `pygame.draw.line()` is tot nu toe niet gebruikt. Zij heeft vijf parameters nodig.

- *Surface* geeft het tekenvlak weer, waarop wordt getekend, in het voorbeeld het `FELD`.
- *Kleur* geeft de kleur van de cirkel aan, in het voorbeeld de voorafgaand gedefinieerde kleur `SCHWARZ`.
- *Beginpunt* geeft het beginpunt van de lijn aan, in het voorbeeld het middelpunt van de klok.
- *Eindpunt* geeft het eindpunt van de lijn aan, in het voorbeeld wordt deze met de functie `punkt()` uit de hoek van de urenwijzer berekend.
- *Dikte* geeft de lijndikte aan.

Dezelfde functie tekent tevens de andere beide wijzers van de klok.

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

Deze regel tekent de minutenwijzer als lijn met een breedte van 4 pixels, vanuit het middelpunt 170 pixels lang in een hoek, die door de minutenwaarde wordt aangegeven.

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

Deze regel tekent de secondewijzer als rode lijn met een breedte van 2 pixels, vanuit het middelpunt 180 pixels lang in een hoek, die door de secondewaarde wordt aangegeven.

`s1 = s` Nu wordt de zojuist weergegeven seconde in de variabele `s1` opgeslagen, om deze waarde in de volgende lusdoorlopen met de actuele seconde te vergelijken.

```
pygame.display.set_caption("Actuele tijd: " +
time.asctime())
```

Deze tijd schrijft de actuele tijd in digitale vorm in het schermonderwerp. Hiervoor wordt de functie `time.asctime()` uit de `time`-bibliotheek gebruikt, die de tijdsaanduiding als voltooid geformatteerde tekenketting levert.

`pygame.display.update()` Tot nu toe zijn alle grafische elementen alleen virtueel getekend. Pas in deze regel wordt de grafische weergave getekend. Het bijwerken gebeurt gelijktijdig. Daarom ziet men ook geen flikkeren bij het na elkaar tekenen van de losse wijzers.

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

Nog altijd binnen de `if`-vraag, en dus een keer per seconden, gebeurt de relatief vermogensverbruikende vraag over eventuele systeemgebeurtenissen, waarmee hier wordt vastgesteld, of de gebruiker binnen de laatste seconde het klokscherm wilde sluiten of op de `[Esc]`-toets heeft gedrukt. Is dit het geval, wordt de variabele `mainloop` naar `False` gezet, en hiermee wordt de lus niet nog een keer gestart.

`pygame.quit()` De laatste regel beëindigt vervolgens de PyGame-module, waardoor tevens het grafische scherm sluit en daarna het volledige programma.

## 10 Grafische dialogvelden voor de programmabesturing

Geen enkel modern programma, dat op enigerlei wijze een interactie met de gebruiker vereist, loopt in zuivere tekstmodus. Overal zijn grafische interfaces, waarop men buttons kan aanklikken, in plaats van invoer via het toetsenbord te moeten uitvoeren.

Python zelf biedt geen grafische interfaces voor programma, heeft echter meerdere externe modules, overeenkomstig de al beschreven PyGame, die er speciaal voor zijn, grafische interfaces te creëren. Een van de meest bekende dergelijke module is *Tkinter*, die de grafische interface *Tk*, die ook voor veel andere programmeertalen kan worden gebruikt, beschikbaar maakt voor Python.

De structuren van de grafische toolkit Tk verschillen enigszins van Python en zien er op het eerste gezicht een beetje apart uit. Daarom beginnen we met een eenvoudig voorbeeld. Een LED moet via buttons in een dialoogveld worden aan- en uitgeschakeld.

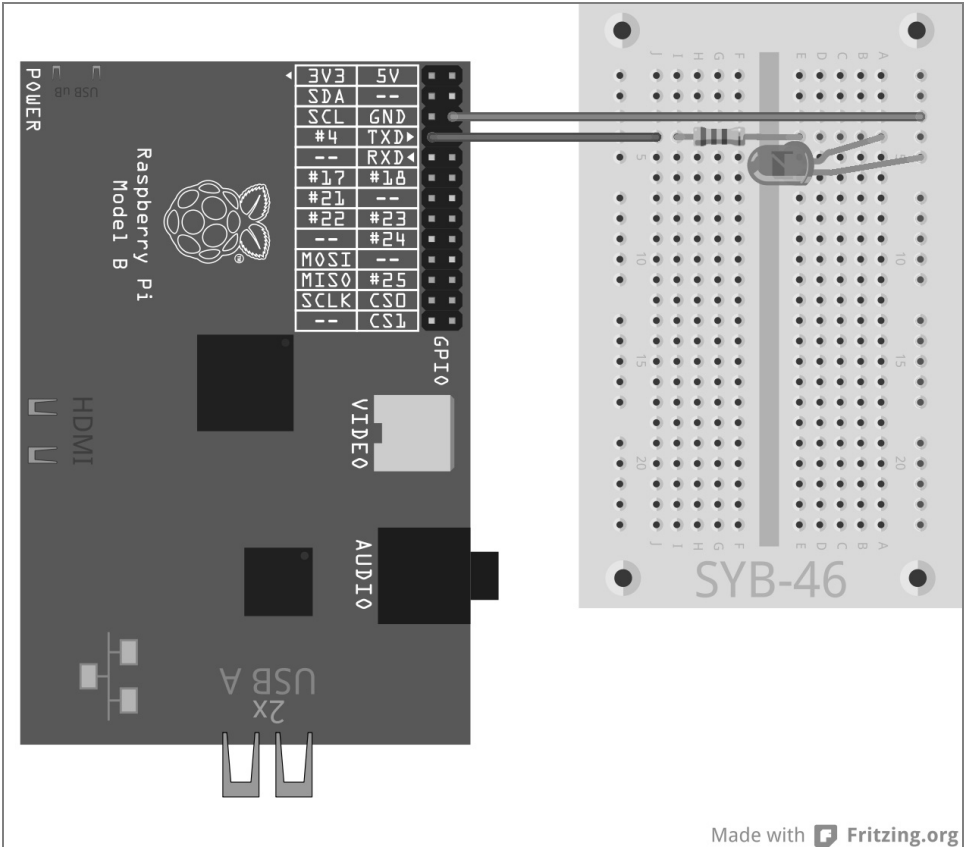
Benodigde onderdelen:

1x insteekprintplaat

1x LED rood

1x 220-ohm-weerstand

2x verbindingskabel



Afb. 10.1: Eén LED aan de GPIO-poort 4.

Sluit een LED via een voorweerstand aan op GPIO-poort 4. Het programma `ledtk01.py` zorgt ervoor dat deze gaan branden.

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

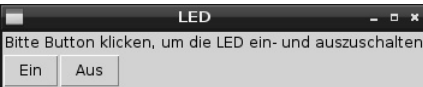
```

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

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

root = Tk(); root.title("LED")
Label(root,
    text="Klik op de toets. om de LED aan- en uit te schakelen").pack()
Button(root, text="Aan", command=LedEin).pack(side=LEFT)
Button(root, text="Uit", command=LedAus).pack(side=LEFT)
root.mainloop()
GPIO.cleanup()

```



**Afb. 10.2:** Zo zal het voltooide dialoogveld er uitzien.

### 10.1.1 Zo werkt het

Dit programma laat de basisfuncties van de Tkinter-bibliotheek zien voor de opbouw van de grafische dialoogvelden. In tegenstelling tot de grafiekbibliotheek PyGame, waarmee grafieken op de pixel nauwkeurig worden opgebouwd, ontstaat de grootte van de dialoogvelden en besturingselementen in Tkinter uit de steeds noodzakelijke grootte automatisch, kan echter naar behoefte ook naderhand handmatig worden beïnvloed.

```

import RPi.GPIO as GPIO
from Tkinter import *

```

Na het importeren van de GPIO-bibliotheek worden aanvullend nog de elementen van de Tkinter-bibliotheek geïmporteerd.

```

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

```

Deze regels laten niets nieuws zien. De GPIO-poort 4 wordt als uitvoerpoort voor een LED gedefinieerd en met de variabele LED gekenmerkt.

```

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

```

Nu wordt een functie `LedEin()` gedefinieerd, die de LED inschakelt.

```

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

```

Een overeenkomstige functie, `LedAus()`, schakelt de LED weer uit. Deze beide functies worden later via de beide toetsen in het dialoogveld opgeroepen.

Tot nu toe was alles alleen maar Python, nu gaan we verder met Tk en zijn eigenaardigheden.

`root = Tk()` Tkinter werkt met zogenaamde widgets. Hierbij betreft het zelfstandige beeldschermelementen, in de meeste gevallen dialoogvelden, die van hun kant verschillende elementen bevatten. Elk programma heeft een `root`-widget nodig, van waaruit alle andere objecten worden opgeroepen. Deze `root`-widget heet altijd `Tk()`, genereert automatisch een scherm en initialiseert ook de Tkinter-bibliotheek.

`root.title("LED")` Objecten in Tkinter stellen verschillende methoden voor vele doeleinden ter beschikking. De methode `title()` in een widget stelt het schermonderwerp in, schrijft dus in dit geval het woord `LED` in de onderwerpregel van het nieuwe scherm.

Elke widget kan meerdere objecten bevatten, die individueel worden gedefinieerd. Tkinter kent hiervoor verschillende objecttypes, waarvan elk verschillende parameters mogelijk maakt, die de eigenschappen van het object beschrijven. De parameters worden, door komma gescheiden, tussen haakjes achter het objecttype aangegeven. Aangezien deze lijst zeer lang kan worden, schrijft men doorgaans elke parameter in een eigen regel, zodat alle parameters onder elkaar zijn uitgelijnd. In tegenstelling tot de inspruingen bij lussen en vragen in Python, zijn deze inspruingen bij Tkinter-objecten echter niet verplicht.

```
Label(root, text="Bitte Button klicken, um die LED ein- und  
auszuschalten").pack()
```

Objecten van het type `Label` zijn zuivere teksten in een widget. Deze kunnen door het programma worden gewijzigd, bieden echter geen interactie met de gebruiker. De eerste parameter in elk Tkinter-object is de naam van de bovenliggende widget, meestal van het scherm, waarin zich het betreffende object bevindt. In ons geval is dat het enige scherm in het programma, de `root`-widget.

De parameter `text` bevat de tekst, die op het label moet worden weergegeven. Aan het einde van de objectdefinitie, wordt de zogenaamde packer als methode `.pack()` bijgevoegd. Deze packer bouwt het object in de dialoogvenster en genereert de geometrie van de widget.

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

Objecten van het type `Button` zijn schakelvlakken die de gebruiker aanklikt, om een bepaalde actie te activeren. Ook hier bevat de parameter `text` de tekst, die op het button moet worden weergegeven.

De parameter `command` bevat een functie, die de button bij het aanklikken oproept. Hierbij kunnen geen parameters worden overgedragen en de functienaam moet zonder haakjes worden weergegeven. Deze button roept de functie `LedEin()` op, die de LED inschakelt.

De methode `.pack()` kan ook nog parameters bevatten, die vastleggen, hoe een object binnen het dialoogveld moet worden toegewezen. `side=LEFT` betekent, dat de button links uitgelijnd en niet in het midden moet worden toegewezen.

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

Volgens hetzelfde schema wordt nog een tweede button aangemaakt, die de LED via de functie `LedAus()` weer uitschakelt.



Nu zijn alle functies en objecten gedefinieerd en het eigenlijke programma kan starten.

`root.mainloop()` Het hoofdprogramma bestaat slechts uit een losse regel. Het start de hoofdloop `mainloop()`, een methode van de `root`-widget. Deze programmalus wacht, tot de gebruiker een van de widgets bedient en hiermee een actie activeert.

Het x-symbool rechtsboven voor het sluiten van het scherm hoeft bij Tkinter niet zelf te worden gedefinieerd. Wanneer de gebruiker de hoofdloop sluit `root`, wordt automatisch de hoofdloop `mainloop()` beëindigd.

`GPIO.cleanup()` Het programma loopt verder naar de laatste regel en sluit de geopende GPIO-poorten.

Na de start van het programma verschijnt een dialoogveld op het beeldscherm. Klik op de button *Aan*, om de LED in te schakelen, vervolgens op *Uis*, om haar weer uit te schakelen.

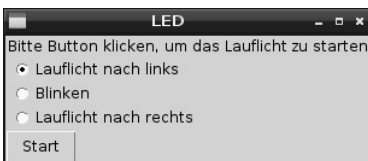
## 10.2 Looplicht met grafisch oppervlak besturen

De Python-bibliotheek Tkinter biedt nog veel meer dan alleen eenvoudige buttons. Via radiobuttons kunnen keuzemenu's worden gebouwd, waarin de gebruiker een van vele aangeboden opties kan kiezen.

### Wat zijn radiobuttons?

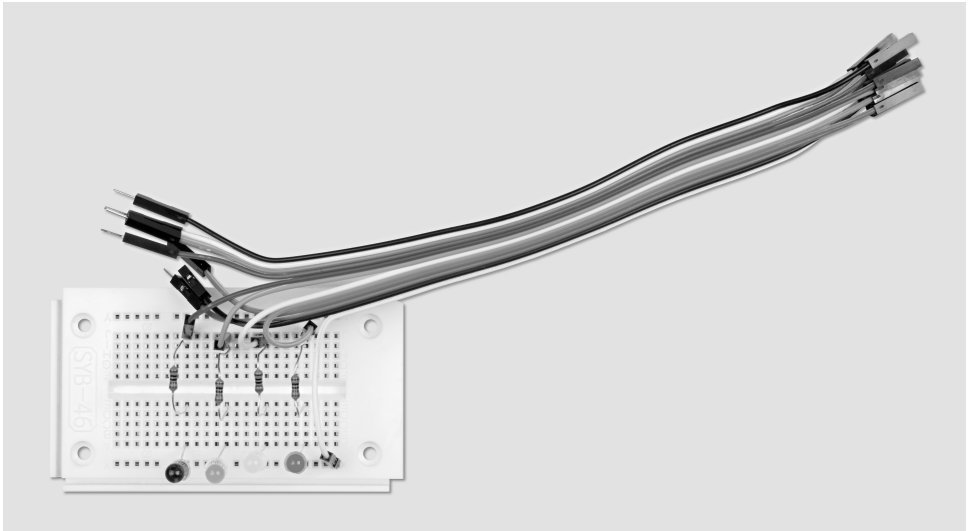
De naam »Radiobutton« komt daadwerkelijk van oude radio's, waarop zendertoetsen voor voorgeprogrammeerde zenders waren aangebracht. Steeds wanneer men een van deze toetsen had ingedrukt, sprong de als laatste ingedrukte door een geraffineerd mechaniek automatisch weer omhoog. Radiobuttons gedragen zich op dezelfde manier. Kiest de gebruiker een optie, worden de anderen automatisch uitgeschakeld.

Het volgende experiment laat verschillende LED-knipperpatronen, die overeenkomen met die uit het experiment »Gekleurde LED-patronen en looplichten«. In tegenstelling hierop, hoeft de gebruiker geen getallen in te voeren op het tekstbeeldscherm, maar kan rustig uit een eenvoudige lijst het gewenste patroon kiezen.



Afb. 10.3: Het dialoogvenster biedt drie LED-patronen om uit te kiezen.

De opbouw van de schakeling is dezelfde als die in het experiment »Gekleurde LED-patronen en looplichten«.



**Afb. 10.4:** Opbouw insteekbord voor experiment 10.2.

**Benodigde onderdelen:**

1x insteekprintplaat

1x LED rood

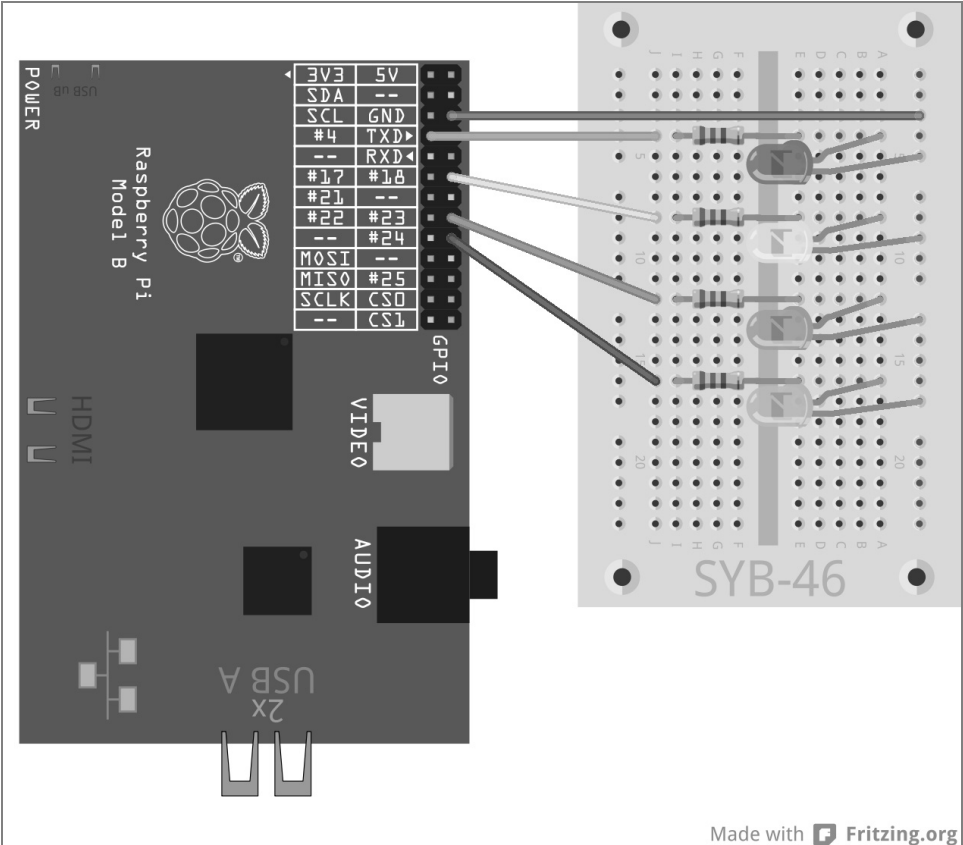
1x LED geel

1x LED groen

1x LED blauw

4x 220-ohm-weerstand

5x verbindingkabel



Made with Fritzing.org

Afb. 10.5: Vier LED's knipperen in verschillende patronen.

Het programma `ledtk02.py` gebaseerd op het vorige programma, is echter met de radiobuttons alsmede met de functies voor de LED-looplichten en knippervoorbeelden uitgebreid.

```
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 = [
    ("Looplicht naar links",1),
    ("Knipperen",2),
    ("Looplicht naar rechts",3)
```

```

]
root = Tk(); root.title("LED"); v = IntVar(); v.set(1)
def LedEin():
    e = v.get()
    if e == 1:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True)
                time.sleep(t)
                GPIO.output(LED[j], False)
    elif e == 2:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True)
                time.sleep(t)
            for j in range(4):
                GPIO.output(LED[j], False)
                time.sleep(t)
    else:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[3-j], True): time.sleep(t)
                GPIO.output(LED[3-j], False)

Label(root,
       text="Klik op de toets, om het looplicht te
       starten").pack()

for txt, m in voorbeeld:
    Radiobutton(root, text = txt,
                variable = v, value = m).pack(anchor=W)
Taste(root, text="Start", command=LedEin).pack(side=LEFT)
root.mainloop()
GPIO.cleanup()

```

### 10.2.1 Zo werkt het

In het begin worden opnieuw de noodzakelijke bibliotheken geïmporteerd. Als aanvulling op het laatste programma is ook de `time`-bibliotheek aanwezig, die nodig is voor de wachttijden bij de LED-knippereffecten.

```

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

```

Aansluitend wordt een lijst gedefinieerd voor de vier LED's. De betreffende GPIO-poorten worden als uitgangen gedefinieerd en op 0 ingesteld, opdat alle LED's in het begin zijn uitgeschakeld.

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

Twee variabelen leggen twee waarden in het programma vast: het aantal herhalingen `w` van een patroon alsmede de knippertijd `t`. Het is mogelijk dat de beide waarden ook bij elk keer dat ze optreden vast in het programma worden ingevoerd. Op deze manier kunnen ze echter gemakkelijker worden aangepast, omdat ze slechts op één plaats zijn gedefinieerd.

```
Muster = [
    ("Looplicht naar links",1), ("Knipperen",2), ("Looplicht naar rechts",3)
]
```

De teksten van de drie patronen, die kunnen worden gekozen, worden in een speciale lijstvorm gedefinieerd. Elk van de drie lijstelementen bestaat uit een waardepaar, elk bestaand uit de aangegeven tekst en een getalwaarde, die later bij de keuze van de betreffende radiobuttons moet worden teruggegeven.

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

De initialisatie van de `root`-widget komt weer overeen met het vorige programma, alleen de inhoud van het dialoogveld is afwijkend.

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

Variabelen, die in Tk-dialoogvelden worden gebruikt, moeten in tegenstelling tot de normale Python-variabelen vóór het eerste gebruik worden gedeclareerd. Deze beide regels declareren een variabele `v` als integer en stellen ze aan het begin op de waarde 1 in.

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

Nu wordt weer een functie gedefinieerd, die net als in het laatste voorbeeld `LedEin()` heet, maar dit keer niet alleen een LED inschakelt, maar een LED-patroon start. De tweede functie `LedAus()` uit het laatste voorbeeld is hier echter niet nodig. De eerste regel van de nieuwe functie leest de gebruikersinvoer van de Tk-variabelen `v` en schrijft de Python-variabele `e`. Hoe de waarde precies in de variabele `v` komt, leest u verderop bij de uitleg van de radiobuttons.

Afhankelijk van de gebruikerskeuze worden drie verschillende programmalussen gestart:

```
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)
```

In het eerste geval loopt een lus vijf keer door, die achtereenvolgend elk van de vier LED's inschakelt, 0,2 seconden laat branden en weer uitschakelt. De vijf herhalingen en de 0,2 seconden knippertijd zijn via de variabelen `w` en `t` aan het begin van het programma gedefinieerd.

```
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)
```

In het tweede geval worden vijf keer achtereenvolgend alle vier LED's gelijktijdig ingeschakeld en, nadat ze gedurende 0,2 seconden hebben gebrand, ook weer gelijktijdig uitgeschakeld.

```

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)

```

Het derde geval is gelijk aan het eerste, met als verschil, dat de LED's achteruit worden geteld en hierdoor het looplicht in omgekeerde richting loopt.

Nadat de functie is gedefinieerd, worden de elementen van de grafische oppervlakken gecreëerd.

```

Label(root,
    text="Klik op de button, om het looplicht te
starten").pack()

```

De tekst van het dialoogveld wordt weer als `Label`-object gedefinieerd. De definitie van de drie radiobuttons is nieuw.

```

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

```

De radiobuttons worden via een bijzondere vorm van de `for`-lus gedefinieerd. In plaats van een lusteller zijn hier twee variabelen aangegeven, die parallel worden geteld. De beide tellers doorlopen achtereenvolgend de elementen van de lijst `patroon`. Hierbij neemt de eerste telvariabele `txt` de eerste waarde van het waardepaar over: de naast de radiobuttons aan te geven tekst. De tweede telvariabele `m` neemt het nummer van het betreffende patroon uit de tweede waarde van elk waardepaar.

De lus maakt op deze manier drie radiobuttons aan `root` is de widget, waarin de radiobuttons liggen. De parameter `text` van een radiobutton, geeft de weer te geven tekst weer, die in dit geval uit de variabelen `txt` wordt gelezen. De parameter `variable` legt een eerder gedeclareerde Tk-variabele vast, waarin na de keuze door de gebruiker de waarde van de gekozen radiobutton wordt ingevoerd.

De parameter `value` legt voor elke radiobutton een getalwaarde vast, die in dit geval uit de variabele `m` wordt gelezen. Wanneer een gebruiker op deze radiobutton klikt, wordt de waarde van de parameter `value` in de onder `variabele` ingevoerde variabele geschreven. Elke van de drie radiobuttons wordt na zijn definitie gelijk met de methode `.pack()` in in het dialoogveld ingebouwd. De parameter `anchor=W` zorgt ervoor, dat de radiobuttons links uitgelijnd onder elkaar worden uitgelijnd.

```

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

```

De toets wordt overeenkomstig het laatste voorbeeld gedefinieerd.

```

root.mainloop(); GPIO.cleanup()

```

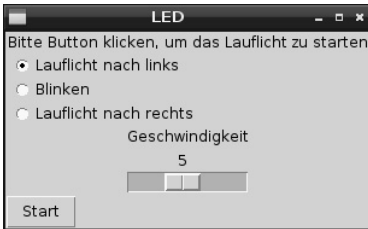
De hoofdloop en het programma-einde komen ook overeen met het laatste voorbeeld.

Start het programma en kies een knipperpatroon via een van de radiobuttons. De eerste keuze is via de variabele `v` vooraf gekozen. Wanneer u radiobuttons gebruikt in een dialoogveld, moet u altijd een nuttige voorkeuze vastleggen, opdat er nooit een ongedefinieerd resultaat ontstaat, mocht de gebruiker zelf geen

keuze maken. Door op *Start* te klikken, start aansluitend het gekozen patroon en laat het vijf keer lopen. Hierna kunt u een ander patroon kiezen.

### 10.3 Knippersnelheid instellen

In de derde stap wordt het dialoogveld opnieuw uitgebreid. De gebruiker kan nu via een schuifregelaar de knippersnelheid instellen.



Afb. 10.6: Keuze uit drie LED-patronen en de instelbare knippersnelheid.

#### Gebruik van schuifregelaars

Schuifregelaars bieden een uiterst intuïtieve methode voor de invoer van getalwaarden binnen een bepaald bereik. Op deze manier bespaart men een plausibiliteitsverzoek, dat bepaalt, of de gebruiker een waarde heeft ingevoerd, die het programma ook doelmatig kan implementeren, omdat waarden buiten het door de schuifregelaar gegeven bereik niet mogelijk zijn. Stel de schuifregelaar altijd zo in, dat de waarden voorstelbaar zijn voor de gebruiker. Het heeft geen zin, waarden in miljoenen te laten instellen. Als de absolute getalwaarde geen belangrijke rol speelt, geef de gebruiker dan een schaal van 1 tot en met 10 of 100 en reken de waarde overeenkomstig om in het programma. De waarden moeten van links naar rechts oplopen, omgekeerd vinden de meeste gebruikers vreemd. Geef bovendien altijd een zinvolle waarde, die wordt overgenomen, wanneer de gebruiker de schuifregelaar niet wijzigt.

Het programma `ledtk03.py` komt grotendeels overeen met het vorige voorbeeld, alleen de regeling van de snelheid wordt aangevuld.

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

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

w = 5
Muster = [
    ("looplicht naar links",1), ("knipperen",2), ("looplicht naar 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="Klik op de button, om het looplicht te
starten").pack()

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

Label(root, text="snelheid").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 Zo werkt het

De initialisatie van de bibliotheken en GPIO-poorten alsmede de definitie van de lijst voor de drie knipperpatronen komen overeen met het voorafgaande programma. De definitie van de variabelen `t` voor de knippertijd vervalt, omdat deze later van de schuifregelaar wordt afgelezen.

`g = IntVar(); g.set(5)` Aanvullend op de Tk-variabelen `v`, waarin het gekozen knipperpatroon wordt opgeslagen, wordt een andere in interger-variabele `g` voor de snelheid gedeclareerd. Deze bevat een startwaarde van 5, die overeenkomt met de gemiddelde waarde van de schuifregelaar.

```

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

```

De functie, die de LED's laat knipperen, komt eveneens overeen met het vorige voorbeeld, echter met een verschil. De variabele `t` voor de knipperduur wordt bepaald uit de waarde van de schuifregelaar.



Omdat een gebruiker intuïtief het sneller knippen combineert met een hogere snelheid, wordt de schuifregelaar naar rechts hogere waarden teruggeven. In het programma is echter voor een hogere snelheid een kortere wachttijd, oftewel een lagere waarde nodig. Dit wordt bereikt door het berekenen van de omgekeerde waarde, die uit de waarden 1 tot en met 10 van de schuifregelaar, de waarden 1.0 tot en met 0.1 voor de variabele  $t$  bepaalt. In de formule moet 1.0 en niet 1 staan, opdat het resultaat een drijvend-kommagetal en geen heel getal wordt.

### Hele getallen omrekenen naar drijvende-kommawaarden

Het resultaat van een berekening wordt automatisch als drijvend-kommagetal opgeslagen, wanneer minimaal één van de waarden in de formule een drijvend-kommagetal is. Zijn alle waarden in de formule hele getallen (integer), wordt het resultaat tevens tot een heel getal ingekort.

De definitie van het label en de radiobuttons in het dialoogveld worden overgenomen uit het voorgaande voorbeeld.

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

Voor de uitleg van de schuifregelaar wordt een ander label in het dialoogveld geschreven. Omdat deze geen parameters in de `pack()` methode bevat, wordt het horizontaal gecentreerd, onder de radiobuttons ingebouwd.

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

De schuifregelaar is een object van het type `Scale`, die evenals alle objecten in dit dialoogveld als eerste parameter `root` bevat. De parameter `orient=HORIZONTAL` geeft aan, dat de schuifregelaar horizontaal ligt. Zonder deze parameters zou hij verticaal staan. De parameters `from_` en `to` geven de begin- en eindwaarden van de schuifregelaar aan. Let hierbij op de schrijfwijze `from_`, omdat `from` zonder laag streepje een gereserveerd woord is in Python voor het importeren van bibliotheken. De parameter `variabele` legt een eerder gedeclareerde Tk-variabele vast, waarin de actueel ingestelde waarde van de schuifregelaar wordt ingevoerd. De beginwaarde wordt uit de bij de variabelendeclaratie vastgelegde waarde, in dit geval 5, overgenomen.

De schuifregelaar wordt met de `pack()`-methode ook weer horizontaal gecentreerd in het dialoogveld ingebouwd.

De overige programmaonderdelen - de *Start*-button, de hoofdloop en het programmaeinde - worden ongewijzigd uit het vorige voorbeeld overgenomen.

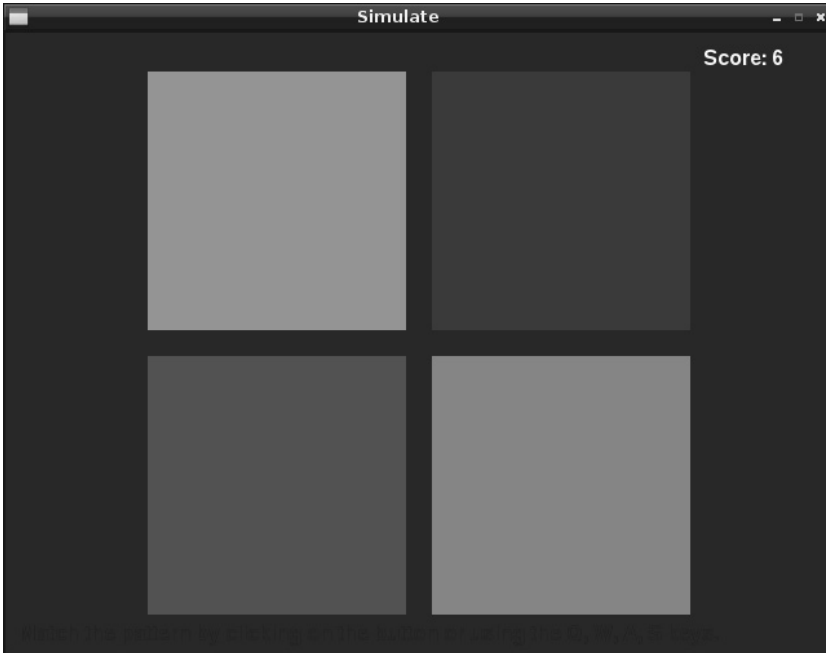
Start het programma, kies een knipperpatroon en leg de snelheid vast. Hogere waarden laten de patronen sneller knippen. Door op de *Start*-button te drukken, leest de functie `LedEin()` het gekozen knipperpatroon uit de radiobuttons alsmede de snelheid uit de positie van de schuifregelaar uit.

## 11 PiDance met LED's

Eind jaren 70, nog voor het tijdperk van de echte computerspellen, bestond er een spel met vier gekleurde lampen, dat in 1979 zelfs op de allereerste keuzelijst van het spel van het jaar stond. het spel was in

Duitsland onder de naam *Senso* op de markt. Atari bracht een kopie onder de naam *Touch Me* ter grootte van een zakrekenmachine uit. Nog een kopie verscheen als *Einstein*, in de Engelstalige landen werd *Senso* als *Simon* op de markt gebracht.

Raspbian levert een grafische versie van dit spel mee bij de *Python Games* onder de naam *Simulate*.

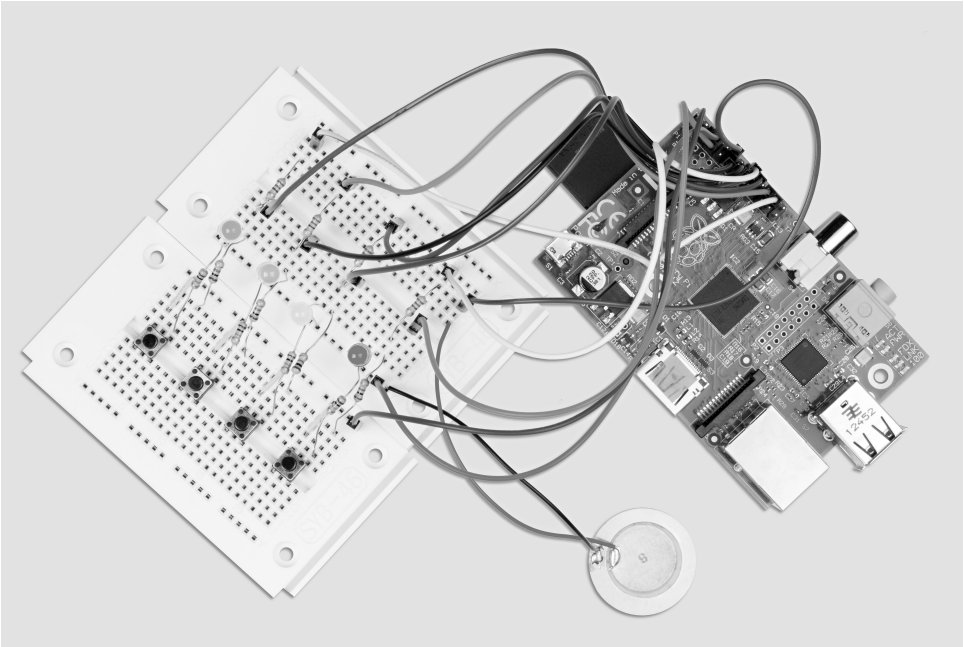


**Afb. 11.1:** Het spel Simulate uit de Python games.

Ons spel PiDance is tevens gebaseerd op dit spelprincipe. De LED's knipperen in een willekeurige volgorde. Aansluitend moet de gebruiker met de toetsen dezelfde volgorde indrukken. Met elke ronde brandt een extra LED, zodat het steeds moeilijker wordt, de volgorde te onthouden. Zodra men een fout maakt, is het spel afgelopen.

Het spel wordt op twee insteekprintplaten gebouwd, zodat de toetsen aan de rand liggen en goed kunnen worden bediend, zonder hierbij ongewild kabels uit de insteekprintplaten te trekken. Voor de betere stabiliteit kunnen de insteekprintplaten aan de lange zijden aan elkaar worden gestoken.

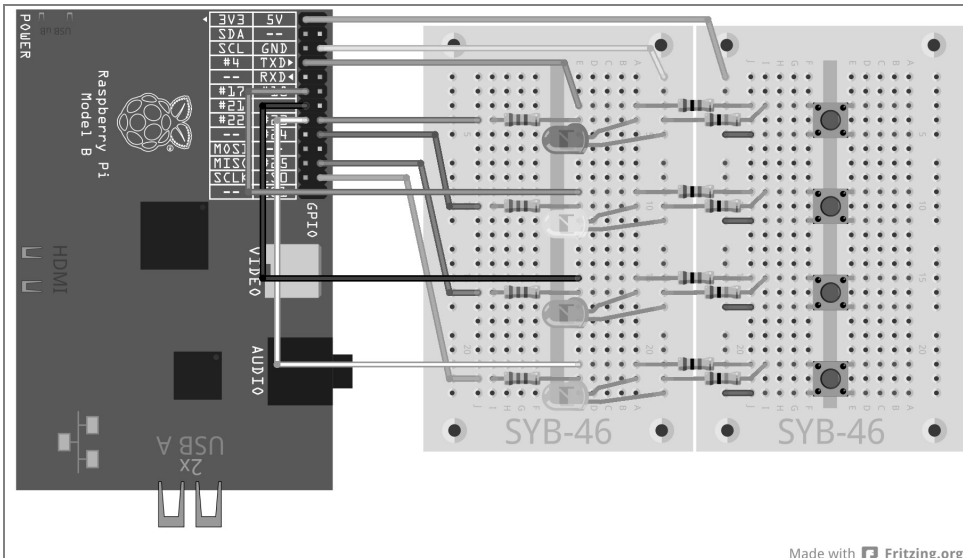
Aanvullend aan de al bekende verbindingkabels zijn nog vier korte draadbruggen nodig. Knip hiervoor met een scherpe tang of een kniptang een ongeveer 2,5 centimeter lang stuk van het meegeleverde staaldraad en verwijder aan beide einden ongeveer 7 millimeter isolatie met een scherp mes. Buig deze draadstukken in een U-vorm. Hiermee kunnen steeds twee rijen op een insteekprintplaat worden verbonden.



**Afb. 11.2:** Opbouw insteekbord voor experiment 11.

**Benodigde onderdelen:**

- 2x insteekprintplaat
- 1x LED rood
- 1x LED geel
- 1x LED groen
- 1x LED blauw
- 4x 220-ohm-weerstand
- 4x 1-kilo-ohm-weerstand
- 4x 10-kilo-ohm-weerstand
- 4x toets
- 10x verbindingkabel
- 4x korte draadbrug



Afb. 11.3: PiDance met LED's en toetsen op twee insteekprintplaten.

De toetsen zijn tegenover de bijbehorende LED's gebouwd. De middelste lange rijen van de insteekprintplaten aan beide zijden van de verbindingsschakel dienen als 0-V- en +3,3-V-leiding voor de schakeling.

Het programma `pidance01.py` bevat het kant-en-klare spel.

```

# -*- coding: utf-8 -*-
import time, random
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
rzahl = 10; kleur = []
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 indrukken():
    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 "Ronde", 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 "Je hebt tot ronde", ronde - 1, "gehaald"
            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 "Geweldig gedaan!"
    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 Zo werkt het

Het programma biedt veel nieuws, de basis van de GPIO-besturing zijn echter bekend.

`rzahl = 10` Na de import van de module `time`, `random` und `Rpi.GPIO` wordt een variabele `rzahl` gecreëerd, die het aantal te spelen ronden vastlegt. U kunt natuurlijk ook meer dan tien ronden spelen - hoe meer ronden, des te moeilijker wordt het, om de knippervolgorde te onthouden.

```

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

```

De lijst `kleur` wordt via een lus met zo veel willekeurige getallen tussen 0 en 3 gevuld, als er ronden worden gespeeld. Hiervoor wordt de methode `append()` gebruikt, die in elke lijst beschikbaar is. Deze is bijgevoegd als parameter doorgezonden element.

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

De GPIO-poorten voor de LED's worden volgens het bekende schema in een lijst `LED` als uitgangen ingesteld en allemaal uitgeschakeld.

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

Volgens hetzelfde principe worden de GPIO-poorten voor de vier toetsen in een lijst `TAST` als ingangen ingesteld.

Hiermee is de basis vastgelegd en worden er nog twee functies gedefinieerd, die meerdere keren nodig zijn in het programma.

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

De functie `LEDein()` schakelt een LED in en laat deze een bepaalde periode branden. De functie gebruikt twee parameters. De eerste parameter, `n`, geeft het nummer van de LED tussen 0 en 3 aan, de tweede parameter, `z`, de periode, die de LED zou moeten branden. Nadat de LED weer is uitgeschakeld, wacht de functie nog 0,15 seconden, tot ze wordt beëindigd, om bij het vaker opvragen van korte pauzes tussen het gaan branden van de LED's te kijken. Dit is vooral belangrijk, wanneer een LED meerdere keren achter elkaar brandt. Anders zou dit niet kunnen worden herkend.

```
def Druucken():
    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
```

De functie `Druucken()` bestaat uit een gesloten lus, die wacht, tot de gebruiker een toets indrukt. Vervolgens wordt het nummer van de toets teruggegeven aan het hoofdprogramma.

`ok = True` Na de definitie van de functies start het eigenlijke hoofdprogramma en stelt als eerste een variabele `ok` in `True`. Zodra de speler een fout maakt, wordt `ok` op `False` ingesteld. Als de variabele na het gegeven aantal ronden nog `True`, heeft de speler gewonnen.

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

Het spel loopt door het in de variabelen `rzahl` vastgelegde aantal ronden. De rondeteller is hierbij 1 naar boven verschoven, opdat het spel in ronde 1 begint en niet in ronde 0.

```
print "Ronde", runde
```

De actuele ronde wordt in het Python-shell-scherf weergegeven.

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

Nu speelt het programma het voorbeeld af, dat de speler moet onthouden. Afhankelijk van het actuele aantal ronden, gaan achtereenvolgens het overeenkomstige aantal LED's branden, volgens de aan het begin van het programma vastgestelde lijst `kleur` met de willekeurig gekozen kleuren. Omdat de teller `ronde` met 1 begint, brandt al in de eerste ronde exact één LED. Om de LED te laten branden, wordt de functie `LEDein()` gebruikt, waarvan de eerste parameter de kleur uit de betreffende lijstpositie is, de tweede parameter laat elke LED gedurende een seconde branden.

`for i in range(ronde):` Nadat het kleurvoorbeeld is afgespeeld, start een andere lus, waarin de speler hetzelfde voorbeeld via de toetsen uit zijn hoofd opnieuw moet invoeren.

`taste = Druucken()` Hiervoor wordt de functie `Druucken()` opgevraagd, die wacht, tot de speler een van de toetsen heeft ingedrukt. Het nummer van de ingedrukte toets wordt in de variabelen `toets` opgeslagen.

`LEDein(toets, 0.2)` Na het indrukken van een toets, gaat de betreffende LED gedurende 0,2 seconde kort branden.

`if(taste != farbe[i]):` Wanneer de als laatst ingedrukte toets niet overeenkomt met de betreffende positie in de lijst, heeft de speler verloren. De operator `!=` staat voor niet gelijk aan. Hier kan ook `<>` worden gebruikt.

```
print "Verloren!"  
print "Je hebt tot ronde", runde - 1, "gehaald"
```

Het programma geeft op het beeldscherm weer, dat de speler heeft verloren en hoeveel ronden hij heeft gehaald. Het aantal voltooide ronden is één lager dan de actuele rondenteller.

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

Als optisch zichtbaar teken worden alle LED's ingeschakeld...

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

... daarna worden ze achtereenvolgens in een afstand van 0,5 seconden weer uitgeschakeld. Zo ontstaat een duidelijk afbouweffect.

`ok = False` De variabele `ok`, de aangeeft, of de speler nog aan het spel deelneemt, wordt op `False` gezet ...

`break` ... in de lus afgebroken. De speler kan geen toetsen meer indrukken. Bij de eerste fout is het direct afgelopen.

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

Wanneer `ok` op `False` staat, wordt tevens de buitenste lus afgebroken, er volgens geen andere ronden meer.

`time.sleep(0.5)` Als de invoer van de volgorde correct was, wacht het programma 0,5 seconden, tot de volgende ronde start.

`if(ok == True):` Het programma bereikt deze plaats, wanneer de lus volledig is doorlopen, de speler dus alle volgordes correct heeft ingevoerd, of de eerdere lus door een speelfout is afgebroken. Indien `ok` nog op `True` staat, volgt de huldiging. In het andere geval wordt dit blok overgeslagen en het spel voert alleen nog de laatste programmaregel uit.

```
print "Geweldig gedaan!"  
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)
```

Bij winst verschijnt een melding in het Python-shell-scherm. Hierna knipperen alle LED's vijf keer kort achterelkaar.

`GPIO.cleanup()` De laatste regel wordt in ieder geval uitgevoerd. Hier worden de gebruikte GPIO-poorten gesloten.



### Impressum

© 2014 Franzis Verlag GmbH, Richard-Reitzner-Allee 2, 85540 Haar bei München

www.elo-web.de

Auteur: Christian Immler

ISBN 978-3-645-10145-5

Alle rechten voorbehouden, waaronder ook fotomechanische reproductie en het opslaan in/op elektronische media. Het maken en verspreiden van kopieën op papier, op gegevensdragers of op het internet, in het bijzonder als PDF, is alleen toegestaan met uitdrukkelijke toestemming van de uitgever en wordt bij gebreke hiervan berecht.

De meeste productaanduidingen van hard- en software alsmede bedrijfsnamen en bedrijfslogo's, die in dit werk worden genoemd, zijn doorgaans gelijktijdig ook gedeponeerde handelsmerken en dienen als dergelijke te worden beschouwd. De uitgeverij volgt bij de productaanduidingen in wezen de schrijfwijzen van de fabrikant.

Alle in dit boek voorgestelde schakelingen en programma's zijn met de grootste zorgvuldigheid ontwikkeld, gecontroleerd en getest.

Desondanks kunnen fouten in het boek en in de software niet volledig worden uitgesloten. De uitgeverij en auteur zijn in het geval van opzet of grove nalatigheid aansprakelijk volgens de wettelijke bepalingen. Verder zijn wij alleen aansprakelijk volgens de Wet op productaansprakelijkheid [Duits: Produkthaftungsgesetz], omwille van schending van leven, lichaam of gezondheid of omwille van opzettelijke schending van wezenlijke contractuele plichten. De claim voor schadevergoeding voor de schending van wezenlijke contractuele plichten is begrensd tot op de contractuele, voorzienbare schade, in zoverre niet een geval van dwingende aansprakelijkheid volgens de Wet op de productaansprakelijkheid aanwezig is.



Elektrische en elektronische producten mogen niet via het normale huisvuil verwijderd worden!

Verwijder het product aan het einde van zijn levensduur conform de geldende wettelijke voorschriften. Voor het retourneren zijn inleverpunten ingericht, waar u elektrische apparaten gratis kunt inleveren. Uw gemeente informeert u, waar u deze inleverpunten kunt vinden.



Dit product is conform de geldende CE-richtlijnen, voor zover u het volgens de meegeleverde gebruiksaanwijzing gebruikt. De beschrijving hoort bij het product en moet worden meegegeven wanneer u het doorverkoopt.

### Let op! Oogbescherming en LED's :

Kijk niet van dichtbij direct in een LED, dit kan netvliesbeschadigingen veroorzaken! Dit geldt in het bijzonder voor heldere LED's in een doorzichtige behuizing, en tevens in bijzondere mate voor power-LED's. Bij witte, blauwe, paarse en ultraviolette LED's geeft de ogenschijnlijke lichtsterkte een verkeerde indruk van het daadwerkelijke gevaar voor uw ogen. Kijk heel goed uit bij het gebruik van focuslenzen. Gebruik de LED's zoals aangegeven in de gebruiksaanwijzing, maar niet met hogere stroom.