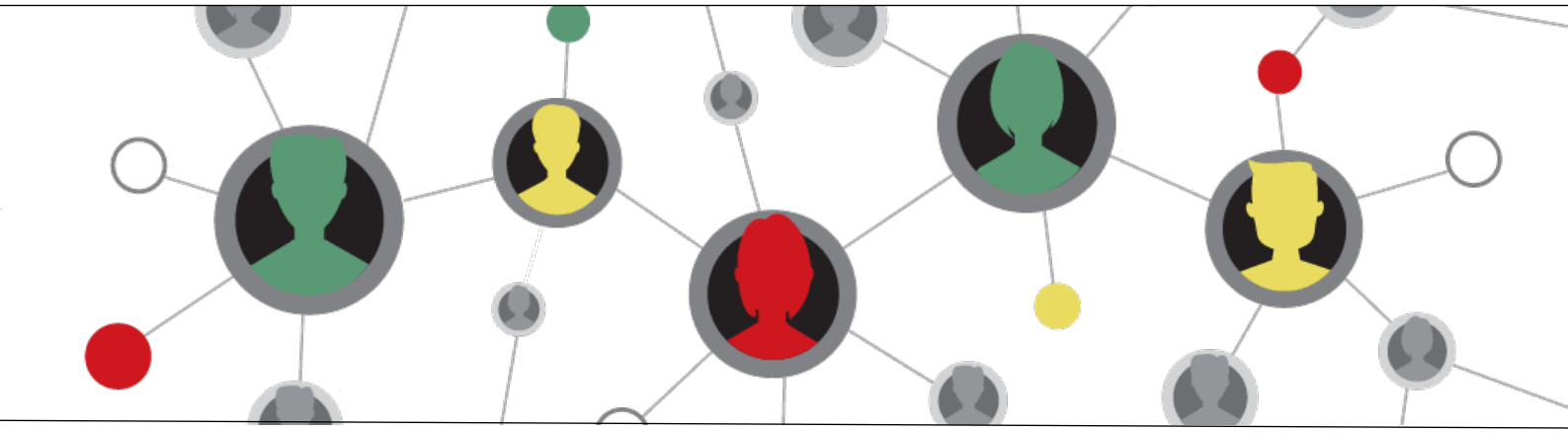
TI Python BootCamp

Python TI-Modules



Bert Wikkerink – Koen Stulens

T³ NEDERLAND

Teachers Teaching with Technology™

T³ VLAANDEREN

TI Python BootCamp

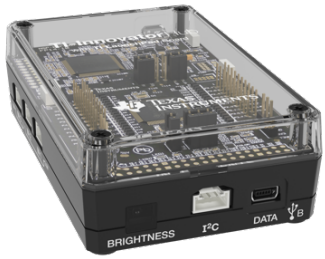
Python TI-modules

Deel 1 Programmeren van een MCU

Deel 2 Data-analyse & grafisch programmeren

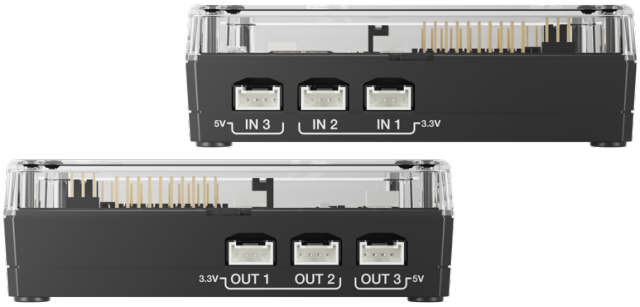
Deel 3 Virtuele STEM

1. TI-Innovator™ Hub

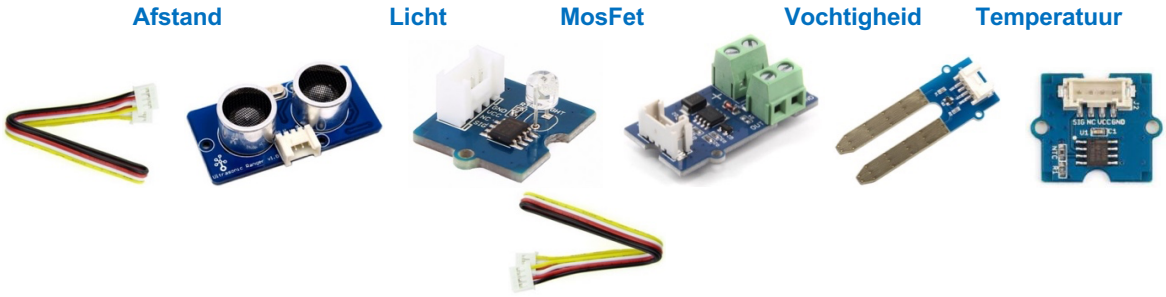


De TI-Innovator is gebaseerd op een evaluatiebord – TI LaunchPad™ – voor de TI-microcontroller MSP432P401R. M.b.v. deze TI LaunchPad-borden testen miljoenen ingenieurs wereldwijd TI MSP432 32 bits microcontrollers voor de ontwikkeling van applicaties.

Door een BoosterPack wordt de TI-Innovator Hub uitgerust met 3 input-poorten en 3 output-poorten voor Grove-apparaten, waaronder tal van sensoren.



Grove apparaten worden ontwikkeld door SeeedStudio, www.seeedstudio.com. Meerdere apparaten/sensoren zijn compatible met voorgeprogrammeerde objecten in TI Python, zoals:



Enkele electronics online stores waar Grove-producten beschikbaar zijn in de Benelux:

- Conrad: www.conrad.be – www.conrad.nl
- KIWI Electronics: www.kiwi-electronics.nl
- DISTRELEC: www.distrelec.be

Bovendien is de TI-Innovator Hub uitgerust met een breadboard connector (om aan de slag te gaan met elektrische circuits, een licht-helderheid sensor en een I²C-poort om randapparatuur aan te sluiten gebruikmakend van het I²C-protocol (zoals de TI-Innovator Rover).

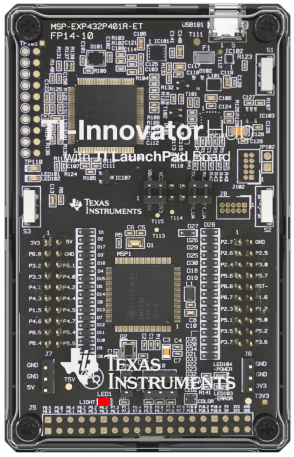
De DATA-poort (mini USB B) wordt gebruikt om de hub te verbinden met TI-handhelds en computers (Windows & Mac). Via de Micro-USB PWR-poort kan extra vermogen toegevoegd worden met b.v. een powerbank.

De TI-Innovator Hub bevat ook een ingebouwde luidspreker om eenvoudig concepten rond geluid en muziek te programmeren en te bestuderen.

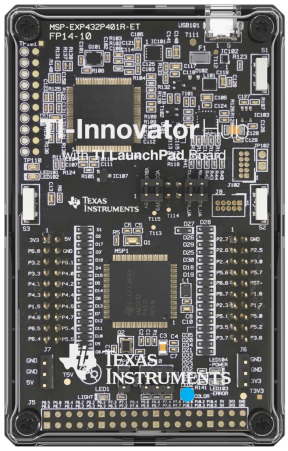
De TI-Innovator Hub is een microcontroller-systeem dat klaar is om op een eenvoudige manier te gebruiken in de klas en om nieuwgierigheid om te zetten in het begrijpen van wetenschappelijke concepten.

De TI-Sketch geïnstalleerd op de microcontroller laat toe om een rode Led en een RGB-Led op de LaunchPad aan te sturen. De TI-Sketch is de software die de communicatie tussen TI-technologie en de TI-Innovator Hub uitvoert via TI Python.

LED

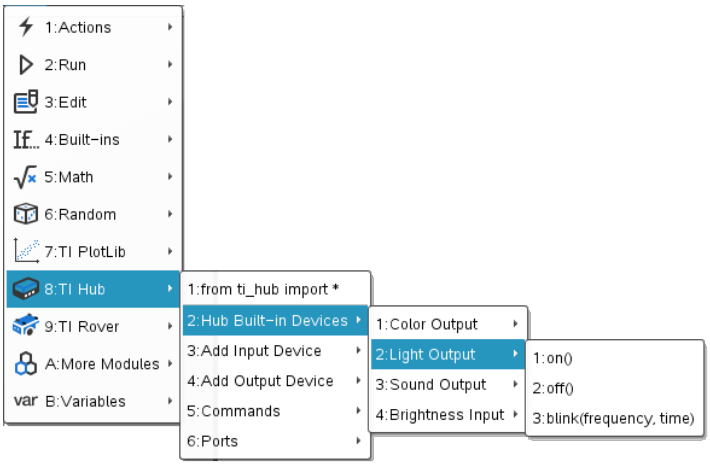


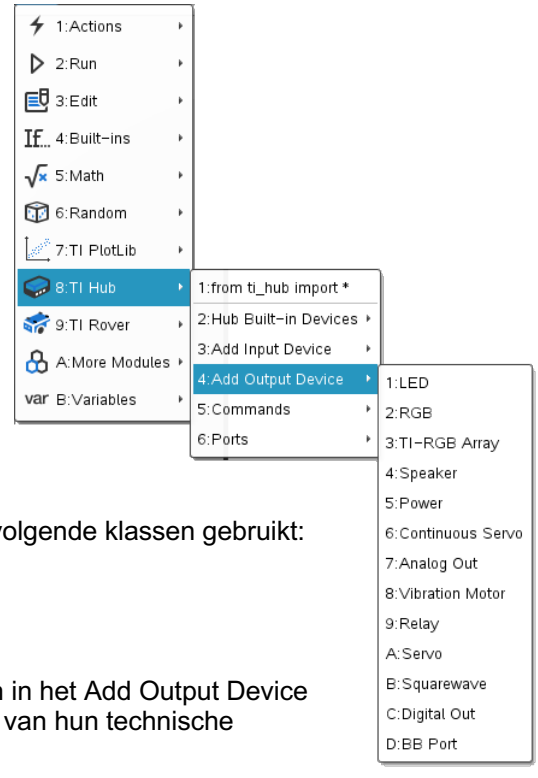
RGB LED



2. Sturen

De module TI Hub bevat tal van voorgedefinieerde klassen voor het aansturen van apparaten. De apparaten zijn ingedeeld in Built-in devices/apparaten en apparaten die aangesloten worden in de output-poorten en de breadboard connector.





Voor het aansturen van de ingebouwde apparaten de, worden de volgende klassen gebruikt:

- **light** voor de rode Led
- **color** voor de RGD-Led
- **sound** voor de luidspreker

Vooraf gedefinieerde klassen voor externe apparaten zijn te vinden in het Add Output Device menu. Apparaten die niet in de lijst voorkomen, kunnen afhankelijk van hun technische specificaties aangestuurd worden met Analog Out en Digital Out.

Merk op dat er maar één instructie tegelijkertijd naar de hub kan gestuurd worden.

De connectie met de hub wordt gemaakt na het uitvoeren van:

```
from ti_hub import *
```

Indien er geen hub verbonden is, krijg je de error-boodschap:

```
tihubException: Hub not connected
```

2.1. Ingebouwde LED

Het statement light.on() zet de rode led op de LaunchPad aan:

```
from ti_hub import *
light.on()
```

Merk op dat zolang de hub aangesloten blijft of een ander programma gerund wordt de rode led aan blijft. Met light.off() wordt de led uitgezet. Met de volgende code

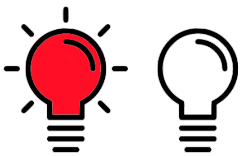
```
from ti_hub import *
light.on()
light off()
```

gaat de led enkel een zeer kleine fractie van een seconde aan daar het off() statement de led uitzet onmiddellijk na het aangaan. Om de led een bepaalde tijd aan te zetten gebruiken we het sleep() statement. Sleep() maakt deel uit van de module ti_hub. Met sleep wordt het sturen van instructies naar de hub tijdelijk stilgezet:

```
from ti_hub import *
light.on()
sleep(1.5)
light off()
```

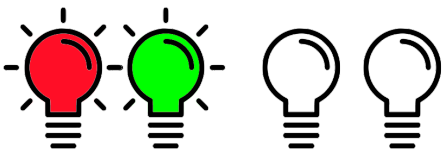
Indien we de led 5-keer aan en uit willen gaan, zetten we de bovenstaande code in een for-lus:

```
from ti_hub import *
for i in range(5):
    light.on()
    sleep(1.5)
    light.off()
    sleep(1.5)
```



Door het statement color.rgb(0,255,0) als volgt toe te voegen aan iedere stap, zal de RGB-led groen kleuren samen met het aangaan van de rode LED. Meer over color in het onderdeel Kleur.

```
from ti_hub import *
for i in range(5):
    light.on()
    color.rgb(0,255,0)
    sleep(1.5)
    light.off()
    color.off()
    sleep(1.5)
```



2.2. Ingebouwde luidspreker

Het statement sound.tone(250,2) laat de frequentie 250 Hz horen voor 2 seconden:

```
from ti_hub import *
sound.tone(250,2)
```

Indien we zoals hieronder een tweede toon toevoegen

```
from ti_hub import *
sound.tone(250,2)
sound.tone(500,2)
```

zal enkel de tweede toon te horen zijn daar de tweede instructie de eerste stopt vooraleer we de eerste toon kunnen horen. Ook hier biedt sleep() een oplossing:

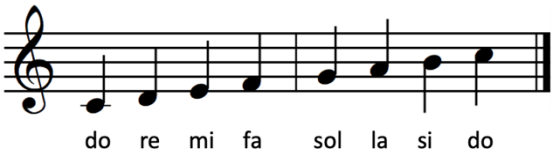
```
from ti_hub import *
sound.tone(250,2)
sleep(2)
sound.tone(500,2)
```

Met een for-lus kunnen we makkelijk een sirene simuleren:

```
from ti_hub import *
for i in range(5):
    sound.tone(500,1)
    sleep(1)
    sound.tone(375,1)
    sleep(1)
```

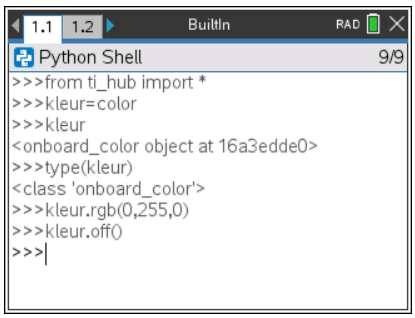
Of een toonladder spelen:

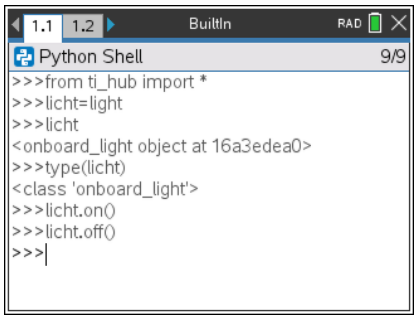
```
from ti_hub import *
ladder=[262,294,330,349,392,440,494,523]
for i in ladder:
    sound.tone(i,1)
    sleep(0.5)
```

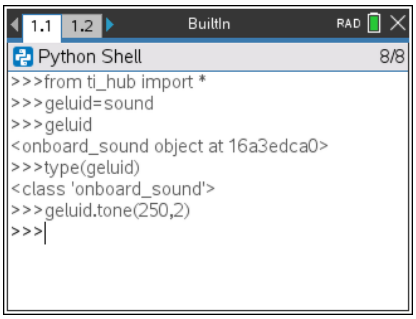


2.3. TI-Innovator Hub interne objecten

Voor de ingebouwde apparaten kunnen we statements uitvoeren zonder eerst een object te definiëren. Voor iedere klasse is er nl. slechts één object/apparaat beschikbaar. Het is echter mogelijk deze objecten te hernoemen en toe te wijzen aan een variabele:

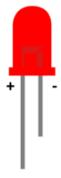






Voor het aansturen van externe apparaten, is het noodzakelijk dat eerst de specifieke objecten gedefinieerd worden.

2.4. Externe LED's

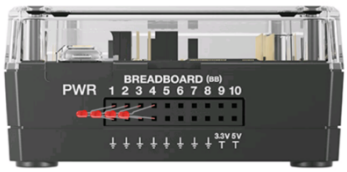


Veronderstel dat we 4 LED's verbinden met de breadboard-poorten BB1 tot en met BB 4. Verbind de lange uiteindes (Anode +) van de LED's in 1 tot en met vier en de korte (Kathode –) in de aarding poorten.

Na het selecteren van de optie 1:LED in het Add Output Device verschijnt de syntax voor het definiëren van een led-object: var=led("port").

Definieer als volgt 4 led-objecten:

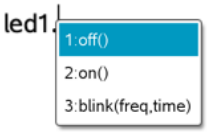
```
from ti_hub import *
led1=led("BB 1")
led2=led("BB 2")
led3=led("BB 3")
led4=led("BB 4")
```



Merk op dat met de cursor in het "port"-veld, automatisch de output-poorten OUT 1, OUT 2 en OUT 3 verschijnen. Deze poorten zijn nodig bij het gebruik van Grove LED-modules.

In het geval dat de LED's rechtstreeks worden aangesloten op de breadboard connector of via een breadboard, dienen de BB-poorten manueel ingegeven worden.

Indien na het definiëren van de LED-objecten, verschijnen de beschikbare methodes (functies) automatisch bij het intikken van een punt na de objectnaam: b.v. **led1.** .

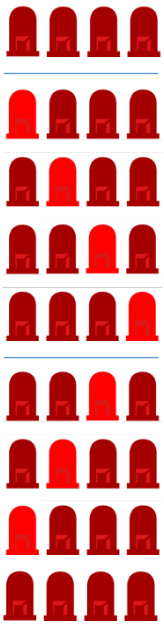


De volgende commando's zetten de vier LEDs aan:

```
led1.on() ; led2.on() ; led3.on() ; led4.on()
```

Met de combinatie van een for-lus en een lijst met de gedefinieerd objecten programmeer je eenvoudig een looplichtje; een voorbeeld van de kracht van de programmeertaal Python:

```
from ti_hub import *
led1=led("BB 1")
led2=led("BB 2")
led3=led("BB 3")
led4=led("BB 4")
ledarray=[led1,led2,led3,led4]
for i in ledarray:
    i.on()
    sleep(0.5)
    i.off()
```



Het terug laten lopen kan met de onderstaande for-lus:

```
for i in range(1,len(ledarray)):
    ledarray[3-i].on()
    sleep(0.5)
    ledarray[3-i].off()
```

2.5. Externe Luidspreker

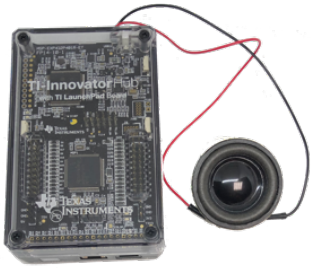
Voor het aansturen van een externe luidspreker, b.v. verbonden met de BB 1-poort, definiëren we het volgende object (Add Output Device > 4:Speaker): luidspreker=speaker("BB 1").

De code voor het spelen van een toon (500 Hz) is gelijkaardig aan de code voor de ingebouwde luidspreker:


```
from ti_hub import *
luidspreker=speaker("BB 1")
luidspreker.tone(500,2)
```

Sirene:

```
from ti_hub import *
luidspreker=speaker("BB 1")
for i in range(5):
    luidspreker.tone(500,1)
    sleep(1)
    luidspreker.tone(375,1)
    sleep(1)
```

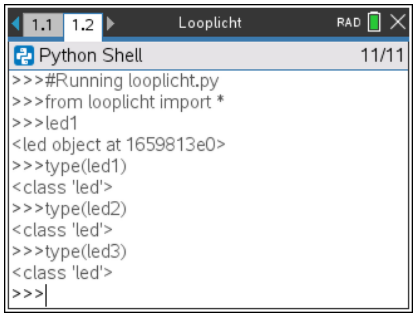


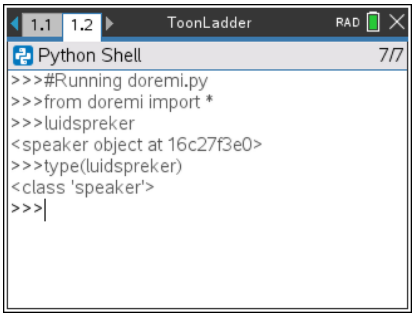
DoReMi:

```
from ti_hub import *
luidspreker=speaker("BB 1")
ladder=[262,294,330,349,392,440,494,523]
for i in ladder:
    luidspreker.tone(i,1)
    sleep(0.5)
```

2.6. TI-Innovator Hub externe objecten

Vanzelfsprekend kan ook voor externe apparaten het type van gedefinieerde objecten bepaald worden:





3. Meten

Het lezen van de waarde van sensoren gebeurt met de object-functie(methode) measurement().

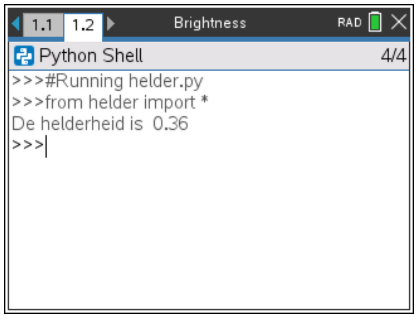
Het meten van de ingebouwde lichthelderheid-sensor gebeurt met de volgende code:

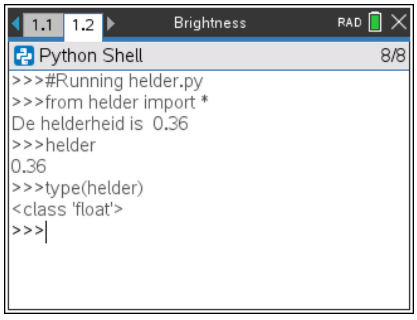
```
from ti_hub import *
brightness.measurement()
```

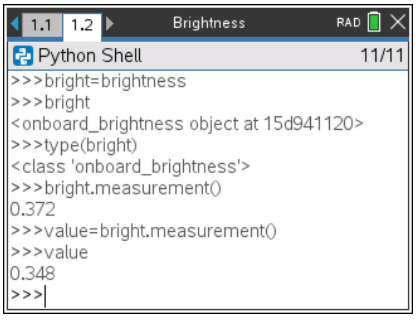
Het runnen van deze code geeft echter geen output. Het weergeven van de gemeten waarde kan als volgt:

```
from ti_hub import *
print("De helderheid is
",brightness.measurement())
```

```
from ti_hub import *
helder=brightness.measurement()
print("De helderheid is ",helder)
```

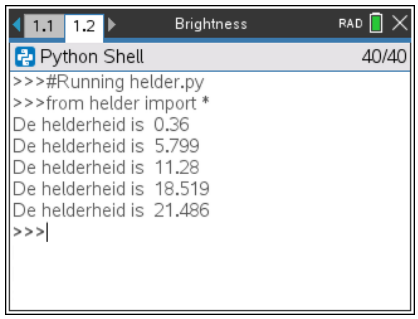






De range van de ingebouwde brightness-sensor gaat standaard van 0 tot 100. In combinatie met een lus kan met de onderstaande code de verschillende helderheid-levels van b.v. de zaklamp van een smartphone bepaald worden:

```
from ti_hub import *
for i in range(5):
    helderheid=brightness.measurement()
    print("De helderheid is ",helderheid)
    sleep(2)
```



Indien we een Grove Light sensor aansluiten op de IN 1-poort gebruiken we de volgende code om de helderheid te bepalen:

```
from ti_hub import *
helderheid=light_level("IN 1")
value=helderheid.measurement()
print("De helderheid is ",value)
```

4. Knipperend led

Periodieke verschijnsels kunnen beschreven worden met behulp van het begrip frequentie, m.a.w. hoe vaak gebeurt iets per seconde. Als alternatief is er de periode; hoe lang één verschijnsel duurt. De eenheid voor frequentie is vernoemd naar Heinrich Hertz, de ontdekker van elektromagnetische golven. De eenheid Hertz (Hz) is het aantal cycli per seconde van een periodiek fenomeen zoals een golf of knipperend licht.

De onderstaande code geeft een visualisatie van kleine frequenties m.b.v. een knipperend led, b.v.

- 1 Hz = 1x aan/uit per seconde
- 2 Hz = 2x aan/uit per seconde
- 4 Hz = 3x aan/uit per seconde

Ingebouwde led

```
from ti_hub import *
f=int(input("Frequentie 1-10 Hz: "))
for i in range(f):
    light.on()
    sleep(1/(2*f))
    light.off()
    sleep(1/(2*f))
```

Externe led in BB 1

```
from ti_hub import *
lamp=led("BB 1")
f=int(input("Frequentie 1-10 Hz: "))
for i in range(f):
    lamp.on()
    sleep(1/(2*f))
    lamp.off()
    sleep(1/(2*f))
```

Voor grotere frequenties wordt het bovenstaande programma onnauwkeurig vanwege o.a. de tijd die nodig is voor het versturen van commando's.

Voor de led-objecten is er ook een functie blink(frequentie,tijd) die het makkelijk maakt om te experimenteren met frequentie en het knipperen van een led. De waarde van frequentie is beperkt tot het bereik 0 – 20 Hz en voor een tijd een bereik van 0.1 – 100 s.

De onderstaande code geeft een gevoel van het verschil tussen 0.5 Hz en 2 Hz:

Ingebouwde led

```
from ti_hub import *
light.blink(0.5,4)
sleep(4)
light.blink(2,4)
```

Externe led in BB 1

```
from ti_hub import *
lamp=led("BB 1")
lamp.blink(0.5,4)
sleep(4)
lamp.blink(2,4)
```

De blink()-functie is ook beschikbaar voor RGB-led's. Hiervoor moet eerst de kleur ingegeven worden voor de blink()-functie te activeren:

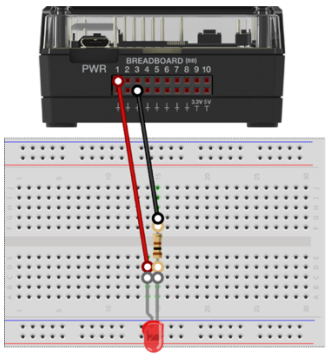
Ingebouwde RGB-led

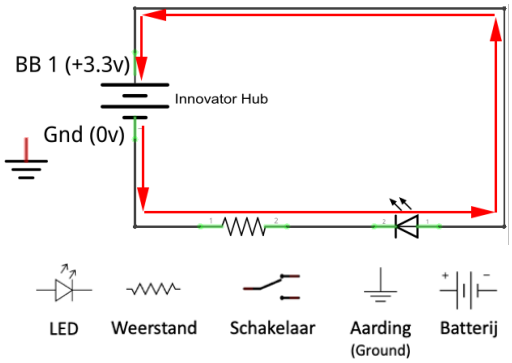
```
from ti_hub import *
color.rgb(150,150,150)
color.blink(0.5,4)
sleep(4)
color(2,4)
```

Externe Grove RGB-led in OUT 1

```
from ti_hub import *
kleur=rgb("OUT 1")
kleur.rgb(150,150,150)
kleur.blink(0.5,4)
sleep(4)
kleur.blink(2,4)
```

Bovenstaande codes, kunnen de start zijn van het bestuderen en sturen van elektrische circuits, gebruikmakend van breadboards: b.v. het experimenteren met weerstanden en het ontdekken van de wet van Ohm: $V = I \cdot R$.





Meer van deze activiteiten op www.wil-destem.be of www.wil-destem.nl.

5. Kleur

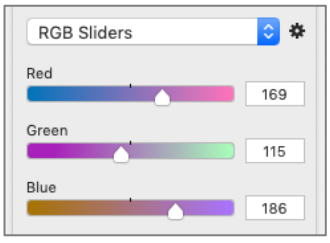
Een RGB-led bestaat uit een combinatie van een rode, groene en blauwe led. Het is mogelijk om praktisch alle kleuren te generen met rood, groen en blauw.

Kleuren produceren met een RGB-led komt neer op het configureren van de intensiviteit van iedere led. Omdat de led's zeer kort bij elkaar staan ziet ons oog niet de individuele kleuren maar enkel de combinatie van de kleuren.

Voor de intensiteit van iedere kleur zijn er $256 = 2^8$ mogelijkheden (8 bits per kleur met waardes van 0 tot 255). In totaal betekent dit 16 777 216 verschillende kleuren.

In 1981 ontwikkelde IBM een 4-bit grafische kaart met 1 byte per kleur, CGA (Color Graphics Adaptor). En voor hun IBM PS/2 computers introduceerde ze in 1987 VGA (Video Graphics Array) gebaseerd op 8 bits per kleur, de start van True Colors op een computerscherm.

Het kleurenpalet van heel wat grafische software is gebaseerd op dit het 8-bit kleuren systeem voor het maken en bewaren van figuren, b.v. PowerPoint.

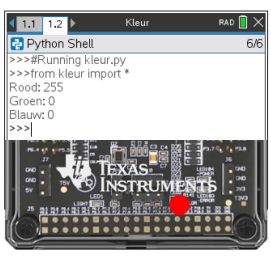


Met de onderstaande code kunnen we experimenteren met de RGB-led:

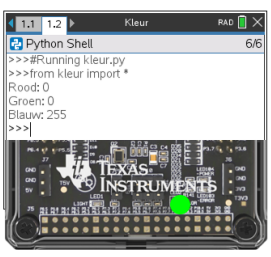
```
from ti_hub import *

r=int(input("Rood: "))
g=int(input("Groen: "))
b=int(input("Blauw: "))

color.rgb(r,g,b)
sleep(2)
color.off()
```

Hieronder enkele voorbeelden:

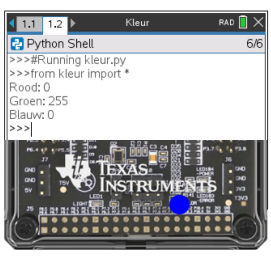
R:255 – G:0 – B:0



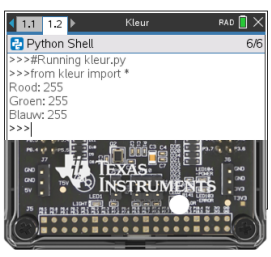
R: – G:255 – B:0



R:0 – G:0 – B:255



R:255 – G:255 – B:255



Gebruikmakend van het randint-statement van de random-module en een loop kunnen we als volgt at random kleuren genereren:

Ingebouwde RGB-led

```
from ti_hub import *
from random import *

while get_key() != "esc":
    r=randint(0,255)
    g=randint(0,255)
    b=randint(0,255)
    color.rgb(r,g,b)
    sleep(1)
    color.off()
    get_key()
```

Grove RGB-LED in OUT 1

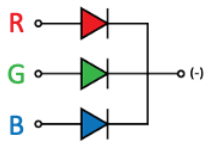
```
from ti_hub import *
from random import *

kleur=rgb("OUT 1")

while get_key() != "esc":
    r=randint(0,255)
    g=randint(0,255)
    b=randint(0,255)
    kleur.rgb(r,g,b)
    sleep(1)
    kleur.off()
    get_key()
```

Indien we een losse RGB-led willen aansturen, connecteren we de RGB-led via de breadboard-connector met de hub.

Hiervoor definiëren we de volgende drie objecten voor de klasse analog_out:


```
from ti_hub import *
from random import *

red=analog_out("BB 1")
green=analog_out("BB 2")
blue=analog_out("BB 3")
```

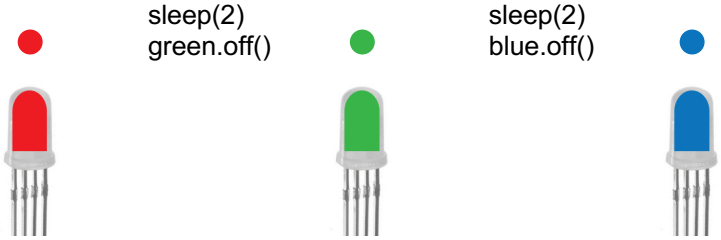
Met de volgende commando's zetten we de led's individueel aan, voor 2 s:




```
red.on()
sleep(2)
red.off()
```

```
green.on()
sleep(2)
green.off()
```

```
blue.on()
sleep(2)
blue.off()
```



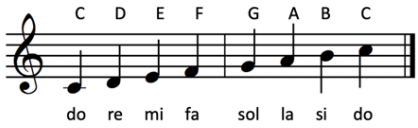
De methode set() laat het toe de intensiteit van iedere led te configureren en zo verschillende kleuren te genereren:

```
t=0
s=int(input("Hoeveel seconden? "))

while t<s:
    r=randint(0,255)
    g=randint(0,255)
    b=randint(0,255)
    red.set(r)
    green.set(g)
    blue.set(b)
    sleep(1)
    t+=1

red.set(0) ; green.set(0) ; blue.set(0)
```

6. Geluid

Voor ons oor is de afstand tussen 110 Hz en 220 Hz gelijk aan de afstand tussen 220 Hz en 440 Hz en tussen 440 Hz en 880 Hz, namelijk een octaaf.

Een octaaf bevat 12 verschillende toonhoogten.



Voor het representeren wordt o.a. gebruik gemaakt van frequenties (Hz) of nootnamen (C3, A#4, F2, ...).

De toonafstand tussen naast elkaar liggende tonen is even groot, met telkens de verhouding $2^{\frac{1}{12}} = 1{,}05946309$.

Hiernaast een tabel met frequenties versus nootnamen.

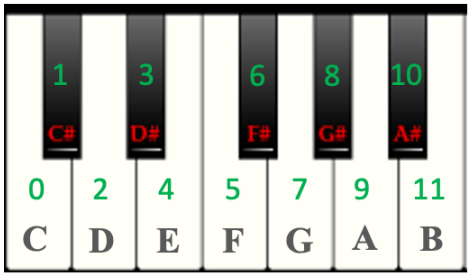
toon	1	2	3	4	5	6
C	32,7	65,4	130,8	261,6	523,2	1046
C#	34,64	69,29	138,5	277,1	554,3	1108
D	36,7	73,41	146,8	293,6	587,3	1174
D#	38,89	77,78	155,5	311,1	622,2	1244
E	41,2	82,4	164,8	329,6	659,2	1318
F	43,65	87,3	174,6	349,2	698,4	1396
F#	46,24	92,49	184,9	369,9	739,9	1479
G	48,99	97,99	195,9	391,9	783,9	1567
G#	51,91	103,8	207,6	415,3	830,6	1661
A	55	110	220	440	880	1760
A#	58,27	116,5	233	466,1	932,3	1864
B	61,73	123,4	246,9	493,8	987,7	1975

Voor A4 = 440 Hz geldt b.v.:

- A#4 is 1 semitoon hoger dan A4: A#4 = $440 \cdot 2^{\frac{1}{12}}$ = 466.1 Hz
- B4 is 2 semitonen hoger dan A4: B4 = $440 \cdot 2^{\frac{2}{12}} =$ 493.8 Hz
- A5 is 12 semitonen hoger dan A4: A5 = $440 \cdot 2^{\frac{12}{12}}$ = 880 Hz

A4 = 440 Hz is de standaard toonhoogte die gebruikt wordt om muziekinstrumenten af te stemmen.

Een andere representaties zijn toonklassen, genummerd van 0 t.e.m. 11.



Het onderstaande programma speelt, na input van de octaaf, een stukje uit *Old MacDonald had a farm*, gebruikmakend van de functie(methode) note(). Voor de hoorbaarheid van de ingebouwde luidspreker best de octaaf niet hoger dan 6.

F F F C D D C
Old MacDonald had a farm
A A G G F
Ee i ee io

F F F C D D C
And on his farm he had some cows
A A G G F
Ee i ee io

```
from ti_hub import *
notes=["F","F","F","C","D","D","C","A","A","G","G","F"]
octaaf=input("Octaaf 1-6: ")
for i in notes:
    sound.note(i+octaaf)
    sleep(0.3)
sleep(1)
for i in notes:
    sound.note(i+octaaf)
    sleep(0.3)
```

Merk op dat het ingeven van het argument tijd voor note() – en tone() – optioneel is.

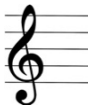
Voor het afspelen van het deuntje via een externe luidspreker, de volgende code:

```
from ti_hub import *
muziek=speaker("BB 1")
notes=["F","F","F","C","D","D","C","A","A","G","G","F"]
octaaf=input("Octaaf 1-6: ")
for i in notes:
    muziek.note(i+octaaf)
    sleep(0.3)
```

De nootnamen voor note() zijn: C, CS (= C#), D, DS (= D#), E, F, FS (= F#), G, GS (G#), A, AS (= A#) en B.

Bovenstaand deuntje kan ook als volgt afgespeeld worden m.b.v. de frequenties:

```
from ti_hub import *
muziek=speaker("BB 1")
notes=[349,349,349,261,293,293,261,440,440,391,391,349]
for i in notes:
    muziek.tone(i+octaaf)
    sleep(0.3)
```



Gebruikmakend van de toonklassen kunnen we de volgende formule afleiden voor de frequenties van de toonhoogtes. Voor de toonklassen nummeren we de noten van 0 tot en met 11, starten met de grondtoon C (do) van de toonladder C-majeur.

C	0	261.60	$\times 2^{\frac{1}{12}}$
CS	1	277.16	$\times 2^{\frac{1}{12}}$
D	2	293.64	
DS	3	311.10	
E	4	329.60	
F	5	349.19	
FS	6	369.96	
G	7	391.96	
GS	8	415.26	
A	9	439.96	
AS	10	466.12	$\times 2^{\frac{1}{12}}$
B	11	493.84	

Frequentie

$$f(n) = 261.60 \cdot 2^{\frac{n}{12}}$$

De formule geldt ook voor $n > 11$ en $n < 0$.
Deze frequenties zijn qua octaaf hoger of lager.

Het spelen van muziek m.b.v. frequenties kan gestuurd worden met frequenties ingevoerd in Lists & Spreadsheet en zo opgeroepen vanuit Python.

	A	B muziek
=		
1	F4	349
2	F4	349
3	F4	349
4	C4	261
5	D4	293
6	D4	293
7	C4	261
8	A4	440
9	A4	440
10	G4	391
11	G4	391
12	F4	349

```
from ti_hub import *
from ti_system import *
muziek=speaker("BB 1")
notes=recall_list("muziek")
for i in notes:
    print(i)
    muziek.tone(i)
    sleep(0.3)
```

De volgende code voor het programmeren van de melodie *Happy Birthday* toont hoe we met nootklassen kunnen gebruikmaken van de functie $f(n) = 261.60 \cdot 2^{\frac{n}{12}}$.

Vermits er enkel numerieke lijsten kunnen uitgewisseld worden tussen TI-Nspire en TI Python declareren we eerst de variabelen c tot en met b zoals hieronder aangeven.

Om de lengte van de noten te bepalen kunnen we b.v. uitgaan van één tel voor een kwartnoot.

	A	B
=		
1	c	0
2	cs	1
3	d	2
4	ds	3
5	e	4
6	f	5
7	fs	6
8	g	7
9	gs	8
10	a	9
11	as	10
12	b	11

B1 c:=0

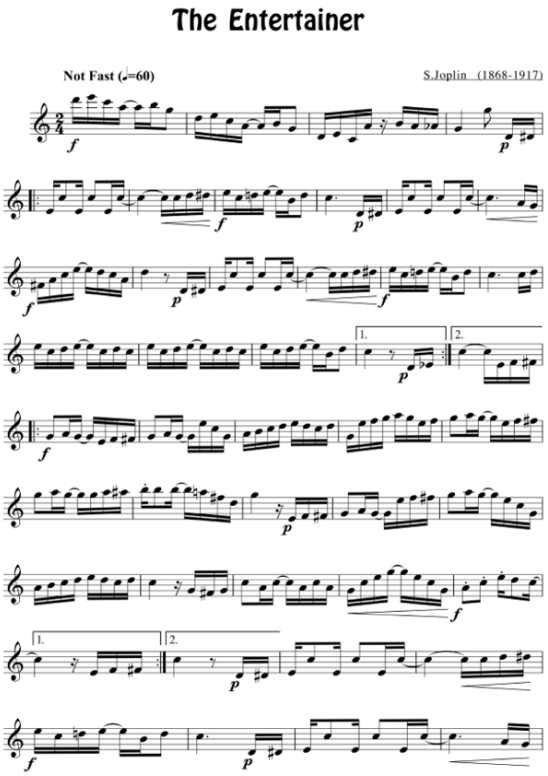
Happy Birthday	toon	tijd
1	d	0.5
2	d	0.5
3	e	1
4	d	1
5	g	1
6	f+1	2
7	d	0.5
8	d	0.5
9	e	1
10	d	1
11	a	1
12	g	2
13	d	0.5
14	d	0.5
15	d+12	1
16	b	1
17	g	1
18	f+1	1
19	e	1
20	c+12	0.5
21	c+12	0.5
22	b	1
23	g	1
24	a	1
25	g	2

Noot	Naam	Lengte	Notatie
	Hele noot	4 tellen	4
	Halve noot	2 tellen	2
	Kwartnoot	1 tel	1
	Kwartnoot met punt	1½ tel	1.5
	Achtste noot	½ tel	0.5

En dan nu de code voor *Happy Birthday*. De variabele tempo bepaalt de snelheid van afspelen.

```
from ti_hub import *
from ti_system import *
bday=speaker("BB 1")
def frequentie(x):
    return 261.6*2**(x/12)
tempo=2
noot=recall_list("noot")
tijd=recall_list("tijd")
for i in range(len(noot)):
    bday.tone(frequentie(noot[i]),tijd[i]/tempo)
    sleep(tijd[i]/tempo)
```

We eindigen met *The Entertainer* van Scott Joplin, gebruikmakend van nootnummers met als referentie C4 = 0.




```
Python Shell                                                    12/12
>>>#Running entertainer.py
>>>from entertainer import *
>>>noot
[26, 28, 24, 21, 21, 23, 19, 14, 16, 12, 9, 9, 11, 7, 2, 4, 0, −3, −3, −1, −3, −4, −5, 7, 2, 3, 4, 12,
4, 12, 4, 12, 12, 12, 14, 15, 16, 12, 14, 16, 12, 14, 12, 2, 3, 4, 12, 4, 12, 4, 12, 9, 7, 6, 9, 12, 16,
16, 14, 12, 9, 14, 2, 3, 4, 12, 4, 12, 4, 12, 12, 12, 14, 15, 16, 12, 14, 16, 12, 14, 12, 12, 14, 16, 1
2, 14, 16, 12, 14, 12, 16, 12, 14, 16, 12, 14, 12, 16, 12, 14, 16, 12, 14, 12]
>>>tijd
[1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 2, 1, 2, 1, 3, 1, 1, 1, 1, 1, 1,
 1, 2, 1, 2, 3, 1, 1, 1, 2, 1, 2, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 2, 1, 2, 1, 3, 1, 1, 1, 1, 1, 1,
 1, 2, 1, 2, 3, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1]
>>>
```

entertainer.py

```
from ti_hub import *
from ti_system import *

muziek=speaker("BB 1")

def frequentie(x):
    return 261.6*2**(x/12)

noot=recall_list("noot")
tijd=recall_list("tijd")

for i in range(len(noot)):
    muziek.tone(frequentie(noot[i]),tijd[i])
    sleep(0.2*tijd[i])
```

7. Zet je code in beweging

In wiskunde en wetenschappen zijn we vaak op zoek naar patronen, modellen en wetmatigheden. In de klas worden tal van algoritmes bestudeerd voor het oplossen van problemen.

Coderen komt er ook op neer algoritmes te programmeren om bepaalde taken automatiseren en samen een oplossing te bieden.

Met de TI-Innovator Rover kunnen we met beweging code visualiseren. Dit maakt het programmeren uitdagender terwijl er gebruik gemaakt wordt van wiskundige en wetenschappelijk concepten.

Zonder in detail te treden van alle mogelijkheden van de TI-Innovator Rover, bekijken we enkele voorbeelden hoe het coderen van de Rover in zijn werk gaat.

Meer info is te vinden op www.education.ti.com/nl of www.education.ti.com/be-nl.

We maken connectie tussen een TI-handheld (TI-Nspire CX II-T (CAS) of TI-84 Plus CE-T PYTHON EDITION) met de code: import ti_rover as rv. Hiermee worden alle methodes(functies) uit de TI Rover-module geïmporteerd voor het object rv.

7.1. Voor- en achteruitrijden

Met de commando's

- rv.forward(1)
- rv.backward(1)

rijdt de Rover 1 unit (10 cm) van het virtuele grid (10 cm) respectievelijk vooruit en achteruit.

Andere eenheden kunnen als volgt toegevoegd worden:

- r.forward(1,"m") 1 meter vooruit
- r.backward(1,"revs") 1 wielomwenteling vooruit

De default-eenheid is grid "units".

7.2. Naar links en rechts draaien

Met de commando's

- rv.left(45)
- rv.right(45)

draait de Rover 45 graden respectievelijk links (tegenwijzerzin) en rechts (wijzerzin).

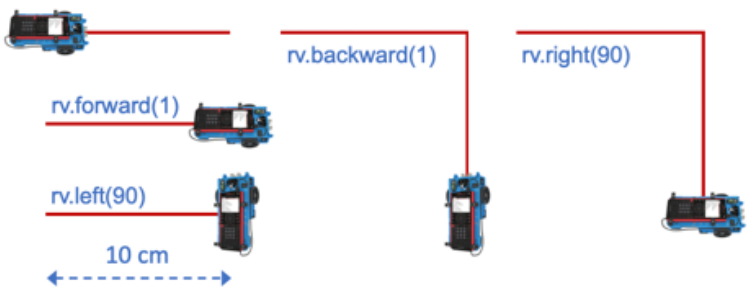
Ook aan deze commando's kan een eenheid toegevoegd worden:

- rv.left(π,"radians") radialen
- rv.right(50,"gradians") 100-delige graden

De default-eenheid is "degrees".

7.3. Een eerste move

```
import ti_rover as rv
rv.forward(1)
rv.left(90)
rv.backward(1)
rv.right(90)
```



7.4. Regelmatige veelhoeken

Een eenvoudige combinatie van deze codes en een for-lus resulteert in het rijden van een pad in de vorm van een regelmatige veelhoek.

Vierkant

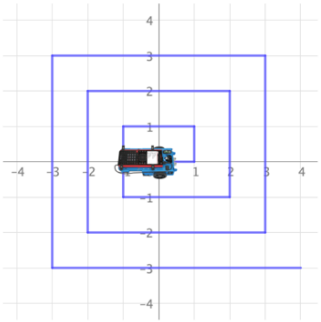
```
import ti_rover as rv

for i in range(4):
    rv.forward(3)
    rv.right(90)
```

Driehoek

```
import ti_rover as rv

for i in range(3):
    rv.forward(3)
    rv.left(120)
```

7.5. Spiralen

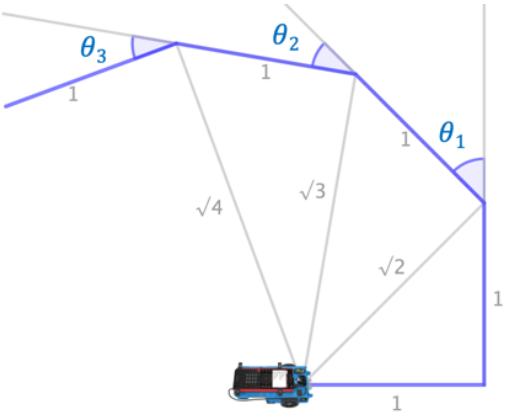
Voorbeeld 1 – Square spiral



```
import ti_rover as rv
for i in range(1,8):
    rv.forward(i)
    rv.left(90)
    rv.forward(i)
    rv.left(90)
```

Voorbeeld 2 – Pythagorean spiral

```
from math import *
import ti_rover as rv
rv.forward(1)
rv.left(90)
rv.forward(1)
for i in range(1,4):
    h=atan(1/sqrt(i))
    rv.left(h,"radians")
    rv.forward(1)
```



$$\theta_1 = \tan^{-1}\left(\frac{1}{\sqrt{1}}\right)$$

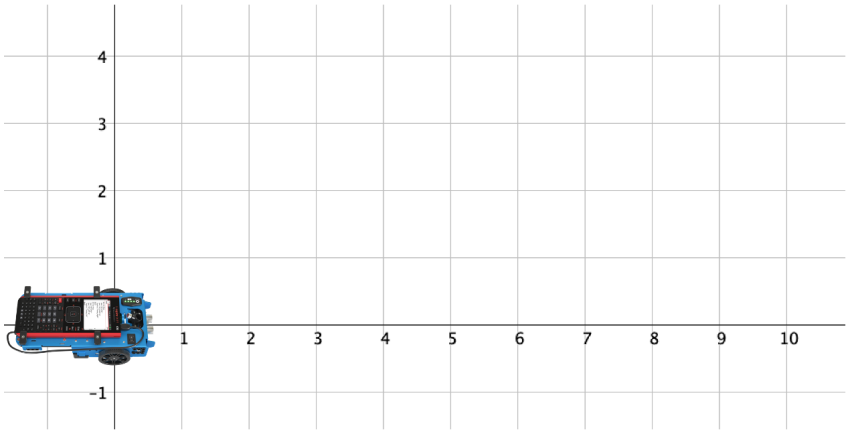
$$\theta_2 = \tan^{-1}\left(\frac{1}{\sqrt{2}}\right)$$

$$\theta_3 = \tan^{-1}\left(\frac{1}{\sqrt{3}}\right)$$

$$\vdots$$

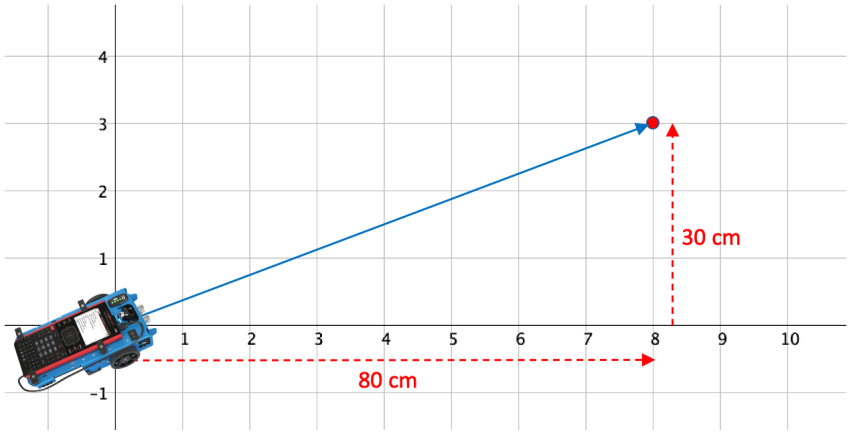
7.6. Rijden via coördinaten

Met de code rv.to_xy(x,y) geef je de Rover de instructie naar het punt met coördinaten (x,y) te rijden in het virtuele grid met de Rover als volgt startend in de oorsprong:

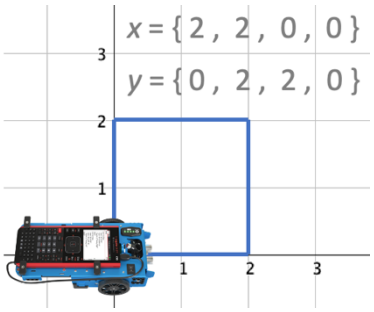


De code import ti_rover as rv laat de Rover eerst gepast draaien en dan naar het punt met coördinaten (8,3) rijden.

```
rv.to_xy(8,3)
```



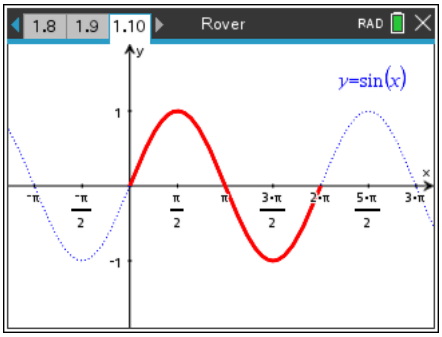
M.b.v. een for-lus rijden we het volgende vierkant:




```
import ti_rover as rv

x=[2,2,0,0]
y=[0,2,2,0]

for i in range(len(x)):
    rv.to_xy(x[i],y[i])
```

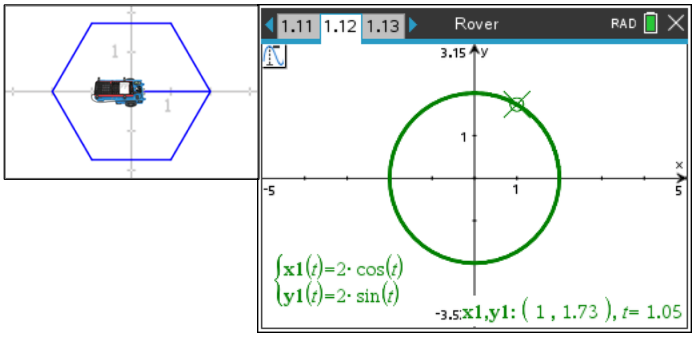
8. Het rijden van functies

Voorbeeld 1 – A sin ride




```
from math import *
import ti_rover as rv

points = 10
scale = 2*pi/points
xcoor=[i*scale for i in range(points+1)]

for i in xcoord:
    rv.to_xy(i,sin(i))
```

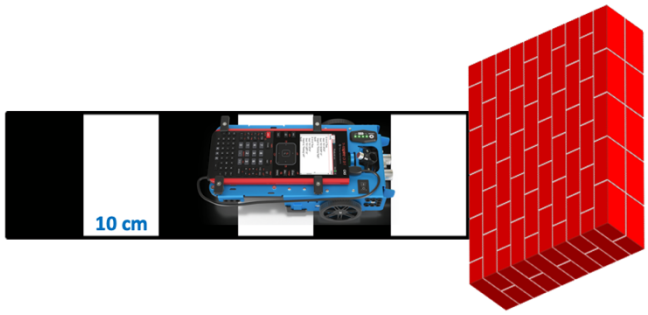
Voorbeeld 2 – In een cirkel rijden




```
from math import *
import ti_rover as rv

points = 6
scale = 2*pi/points
parameter=[i*scale for i in range(points+1)]

for t  in parameter:
    x=2*cos(t)
    y=2*sin(t)
    rv.to_xy(x,y)
```

9. Metend rijden

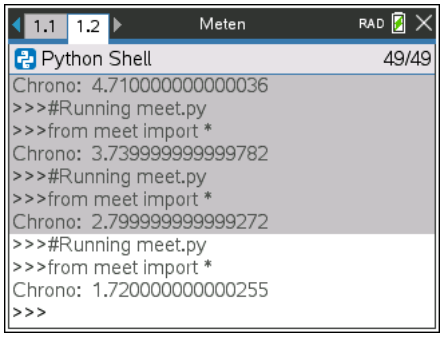
Aan de voorkant van de Rover is een ranger bevestigd waarmee de afstand (in meter) van de Rover tot een object gemeten wordt. Indien de rover te dicht komt met b.v. een muur brengt de methode stop() de Rover tot stilstand.

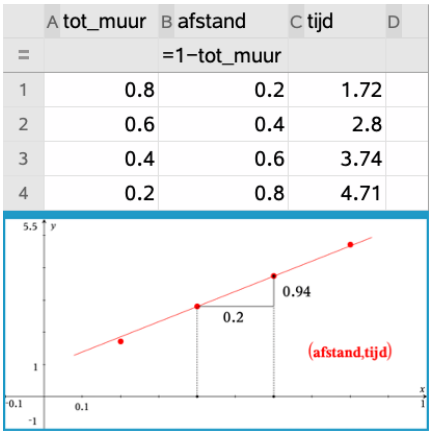



```
import ti_rover as rv
afstand=rv.ranger_measurement()
rv.forward(100)
while afstand > 0.1:
    afstand=rv.ranger_measurement()
rv.stop()
```

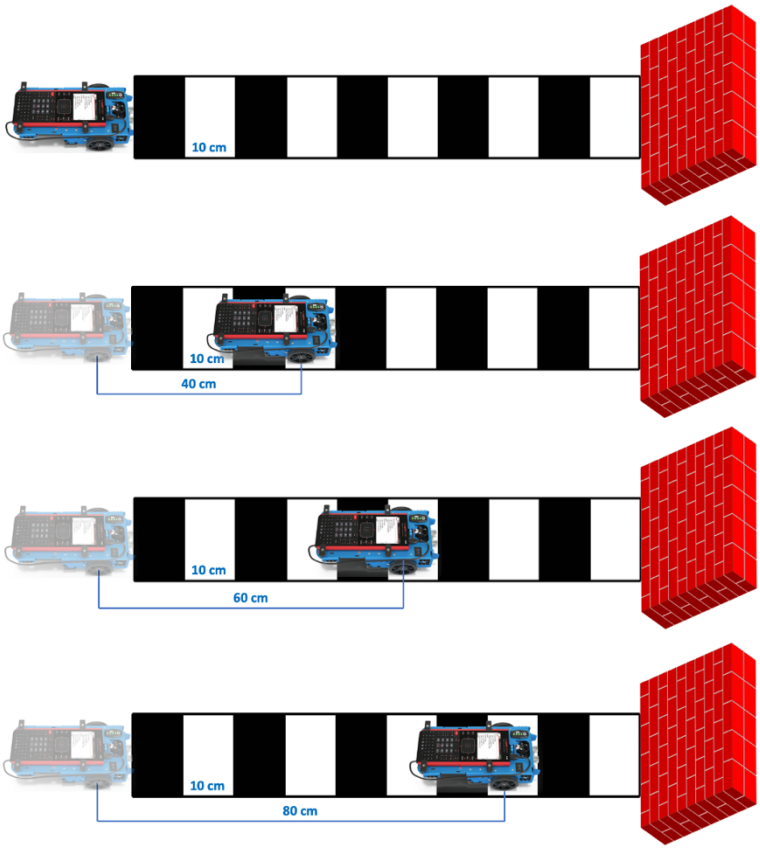
Het volgende programma berekent een ruwe benadering van de snelheid van de eenparig rechtlijnige beweging van de rover. Het statement clock() van de time-module geeft de processor tijd als een float-getal, uitgedrukt in seconden.

```
from time import *
import ti_rover as rv

afstand=rv.ranger_measurement()

start=clock()
rv.forward(100)

tot_muur=0.6

while afstand > tot_muur:
  afstand=rv.ranger_measurement()

rv.stop()
einde=clock()

print("Chrono: ",einde-start)
```





	A tot_muur	B afstand	C tijd	D
=		=1−tot_muur		
1	0.8	0.2	1.72	
2	0.6	0.4	2.8	
3	0.4	0.6	3.74	
4	0.2	0.8	4.71	



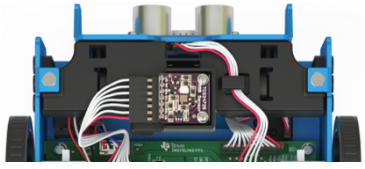
2.80 seconden

3.74 seconden

4.71 seconden

Het experiment geeft een de benadering $v = \frac{0.2}{0.94} = 0.21\,\frac{m}{s}$ van de standaard snelheid van de Rover: $0.2\,\frac{m}{s}$.

De snelheid van de Rover kan varieëren tussen $0.14\,\frac{m}{s}$ en $0.23\,\frac{m}{s}$.



De Rover beschikt ook over een kleuren-sensor waarmee bepaald wordt wat de kleur is waarover de Rover rijdt; Deze kleuren-sensor ledit ook tot uitdagende programmeeropdrachten.

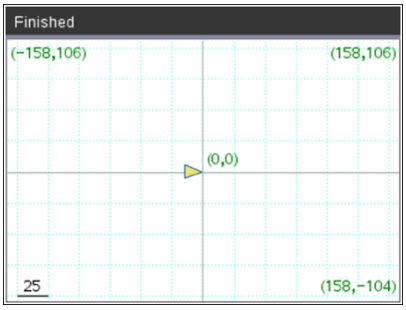
10. Rover versus Turtle

Met de Python Turtle-module kunnen we de Rover voor de voorgaande TI Python programma's simuleren, uitgezonderd deze i.v.m meten. Download en installeer de Turtle-module:

- education.ti.com/nl/product-resources/turtle-module/nspire-python of
- education.ti.com/nl-be/product-resources/turtle-module/nspire-python.

Het runnen van code gebruikmakend van de Turtle-module kan enkel voor de Handheld Page Size.

De Turtle-module maakt gebruik van de volgende scherminstellingen: (xmin,xmax) = (-158,158) en (ymin,ymax) = (-104,106).



Het beginnen met tekenen start met het aanmaken van een Turtle-object:

```
from turtle import *
rv=turtle()
```

Bij start van een programma bevindt rv zich in de oorsprong. Door het toevoegen, na het definiëren van het Turtle-object, van de volgende commando's

```
rv.hideturtle()
rv.hidegrid()
```

verbergen we de Turtle en het grid(assen en schaal inclusief). Voor de output van de onderstaande programma's werden de Turtle en het grid verborgen; zonder de commando's telkens in de code te vermelden.

Voor het simuleren van de Rover-programma's maken we gebruik van de volgende gelijkaardige code:

- rv.forward(distance) rv.backward(distance)
- rv.left(angle_degrees) r.right(angle_degrees)
- rv.clear() wist alle gemaakte tekeningen.
- rv.penup() en rv.pendown() laten toe rv te bewegen zonder te tekenen.
- rv.pencolor(r,g,b) en rv.pensize(value 1, 2 of 3) brengen wat kleur en variatie in het tekenen.

Enkele programma's

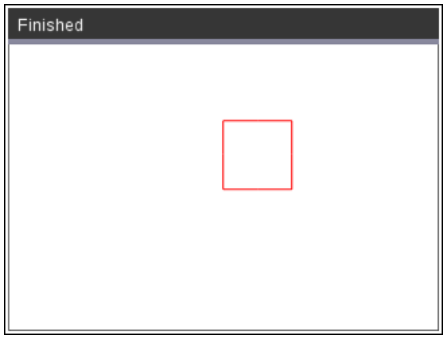
TI-Innovator Rover

Vierkant

```
import ti_rover as rv
for i in range(4):
    rv.forward(3)
    rv.right(90)
```

Turtle Rover

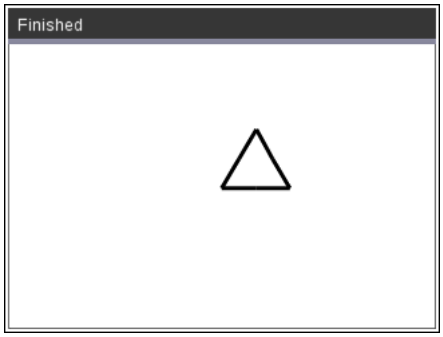
```
from turtle import *
rv=Turtle()
rv.pencolor(255,0,0)
for i in range(4):
    rv.forward(50)
    rv.left(90)
```



Driehoek

```
import ti_rover as rv
for i in range(3):
    rv.forward(3)
    rv.left(120)
```

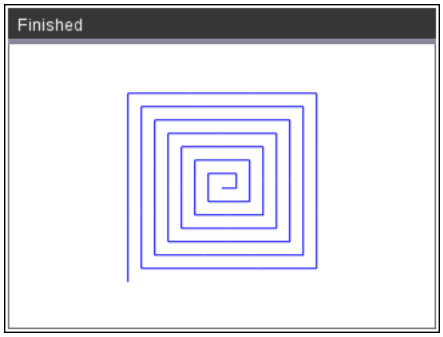
```
from turtle import *
rv=Turtle()
rv.pensize(2)
for i in range(3):
    rv.forward(50)
    rv.left(120)
```



Square spiral

```
import ti_rover as rv
for i in range(1,8):
    rv.forward(i)
    rv.left(90)
    rv.forward(i)
    rv.left(90)
```

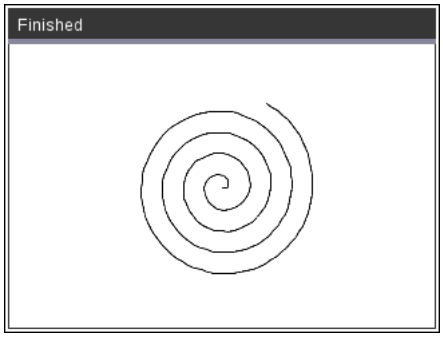
```
from turtle import *
rv=Turtle()
rv.pencolor(0,0,255)
for i in range(1,15):
    rv.forward(i*10)
    rv.left(90)
    rv.forward(i*10)
    rv.left(90)
```



Pythagorean Spiral

```
from math import *
import ti_rover as rv
rv.forward(1)
rv.left(90)
rv.forward(1)
for i in range(1,4):
    h=atan(1/sqrt(i))
    rv.left(h,"radians")
    rv.forward(1)
```

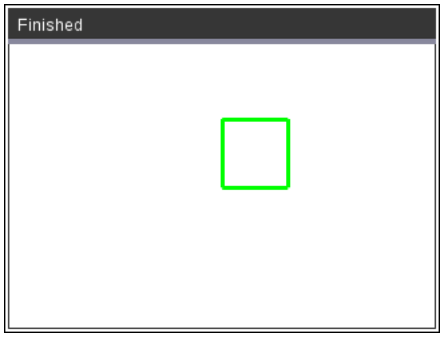
```
from turtle import *
from math import *
rv=Turtle()
rv.forward(5)
rv.left(90)
rv.forward(5)
for i in range(1,200):
    h1=1/sqrt(i)
    h2=atan(h1)*180/pi
    rv.left(h2)
    rv.forward(5)
```



Vierkant xy

```
import ti_rover as rv
x=[2,2,0,0]
y=[0,2,2,0]
for i in range(len(x)):
    rv.to_xy(x[i],y[i])
```

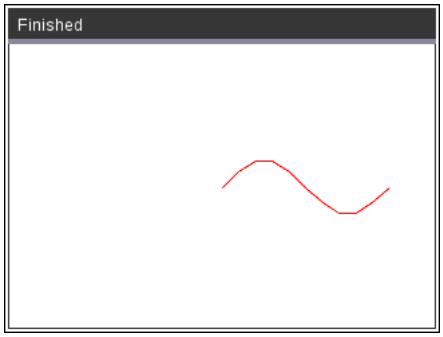
```
from turtle import *
rv=Turtle()
rv.pencolor(0,255,0)
rv.pensize(2)
x=[50,50,0,0]
y=[0,50,50,0]
for i in range(len(x)):
    rv.goto(x[i],y[i])
```



A sin ride

```
from math import *
import ti_rover as rv

points=10
scale=2*pi/points
x=[i*scale for i in
range(points+1)]

for i in x:
    rv.to_xy(i,sin(i))
```

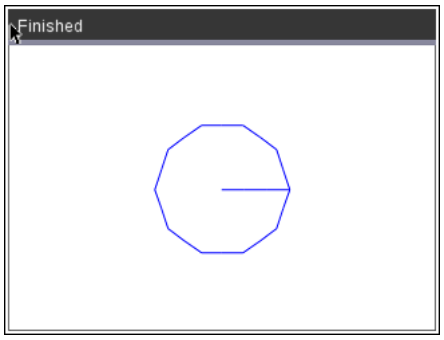
```
from turtle import *
from math import *

rv=Turtle()
rv.pencolor(255,0,0)

points=10
scale=2*pi/points

x=[i*scale for i in
range(points+1)]

for i in x:
    rv.goto(20*i,20*sin(i))
```



In een cirkel rijden

```
from math import *
import ti_rover as rv

points = 6
scale = 2*pi/points
p=[i*scale for i in
range(points+1)]

for t  in parameter:
    x=2*cos(t)
    y=2*sin(t)
    rv.to_xy(x,y)
```

```
from turtle import *
from math import *

rv=Turtle()
rv.pencolor(0,0,255)

points=10
scale=2*pi/points

p=[i*scale for i in
range(points+1)]
for t  in p:
    x=50*cos(t)
    y=50*sin(t)
    rv.goto(x,y)
```



Programmeeropdrachten

Opdracht 1: SOS

a. Programmeer de SOS-code als drie korte tonen gevolgd door drie lange tonen en nogmaals dezelfde drie korte tonen; allemaal in dezelfde toonhoogte. Zend de SOS-code 10-maal na mekaar uit.

b. Combineer de audio SOS-code met het knipperen van een led: drie korte flitsen, drie lange en opnieuw drie korte.

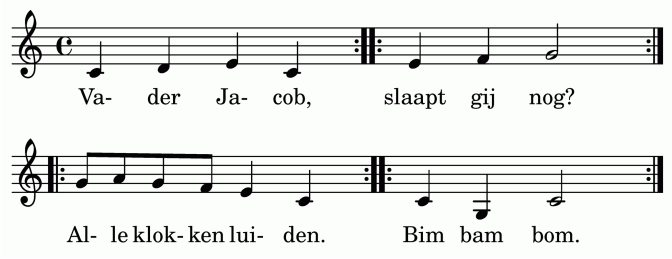
Opdracht 2: Straatverlichting

Schrijf een programma dat een led aanzet indien het duister wordt, b.v. de ingebouwde kleur-led met rgb-waarde wit = (255,255,255), en uitzet indien er terug opnieuw voldoende licht is.

Opdracht 3: Licht-muziek

Codeer het volgende deuntje gebruik makend van letternamen en van frequenties.

C D E C
Broeder Jacob
C D E C
Broeder Jacob
E F G
Slaap jij nog
E F G
Slaap jij nog
G A G F E C
Alle klokken luiden
G A G F E C
Alle klokken luiden
CGC
Bim Bam Bom
CGC
Bim Bam bom



Pas de code aan zodat bij iedere noot de RGB-led in een bepaalde kleur oplicht.

Ga creatief aan de slag met muziek!

Opdracht 4: Rij/Wandel de grafiek

a. Cirkelbeweging

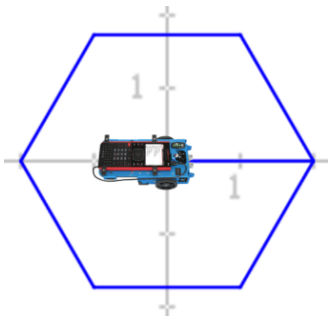
Voor de onderstaande code krijgen we de volgende resultaten.

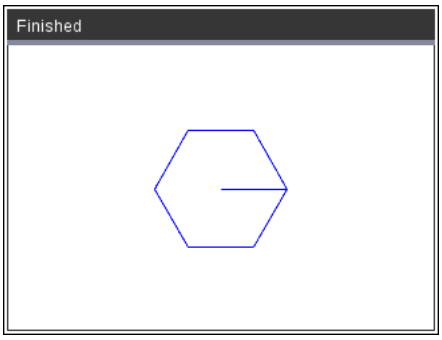
TI-Innovator Rover

```
from math import *
import ti_rover as rv

points = 6
scale = 2*pi/points
p=[i*scale for i in range(points+1)]

for t  in parameter:
    x=2*cos(t)
    y=2*sin(t)
    rv.to_xy(x,y)
```

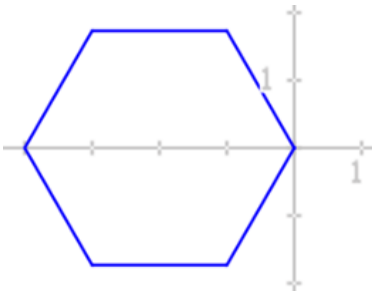
CX Turtle

```
from turtle import *
from math import *
rv=Turtle()
rv.hideturtle()
rv.hidegrid()
rv.pencolor(0,0,255)
points=6
scale=2*pi/points
p=[i*scale for i in range(points+1)]
for t  in p:
    x=50*cos(t)
    y=50*sin(t)
    rv.goto(x,y)
```





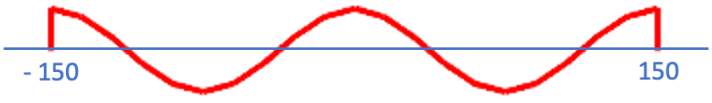
Pas de code voor de parameterkromme aan om de volgende output te krijgen.



b. SinRide

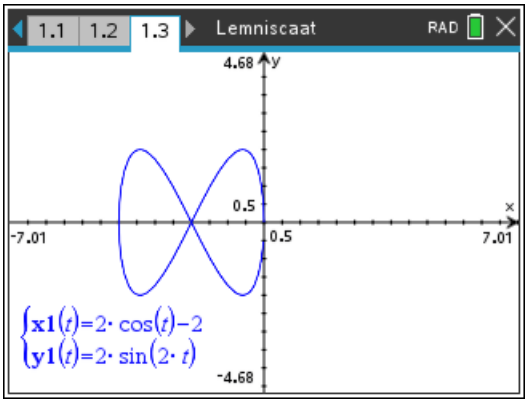
met fase-verschil van grootte $\frac{\pi}{2}$

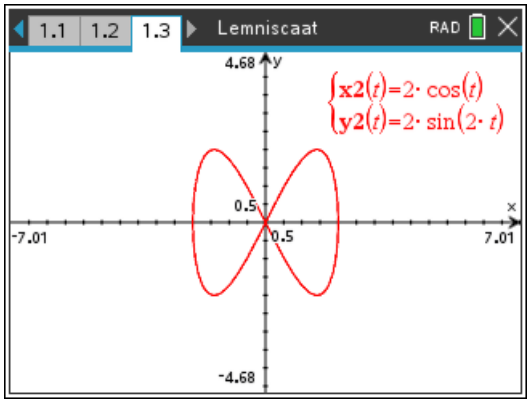
- Laat de Rover een cosinus-curve rijden/tekenen startend in de oorsprong tot 4π.
- Teken de volgende cosinus curve met de Turtle-module met amplitude 20 pixels.



c. Oneindige lus

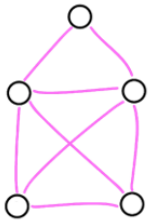
Laat de Rover een oneindige lus rijden/tekenen:





Voor de Turtle-module geldt dat 1 eenheid gelijk is aan 1 pixel. Voor het tekenen van een oneindig lus met de Turtle best een schaalvergroting toevoegen, b.v. 30 in de x-richting en 20 in de y-richting.

d. Eenrichtingsverkeer

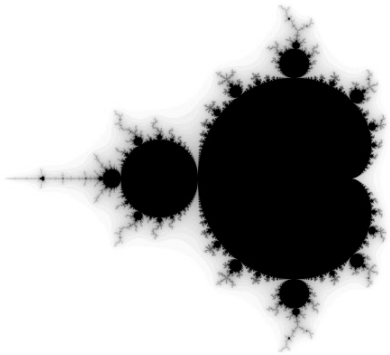


In grafentheorie noemt met het hiernaast afgebeelde huisje een Euler-pad. Dit wil zeggen dat deze figuur in één pennentrek kan getekend worden, van punt naar punt, zonder één lijnstuk dubbel te tekenen.

Schrijf een programma dat de Rover (of de Turtle-module) zo'n huisje laat tekenen:

- gebruikmakend van coördinaten,
- zonder gebruik te maken van coördinaten.

e. Mandelbrot-verzameling



Het hart van de Mandelbrot-verzameling, waar oneindig veel bollen aangehecht zijn, heeft als rand de vorm van een cardioïde.

De vergelijking van een cardioïde, $a \in \mathbb{R}$:

- Parametervoorstelling

$$x(t) = 2a \cdot \cos(t)\,(1 - \cos(t))$$
$$y(t) = 2a \cdot \sin(t)\,(1 - \cos(t))$$

- Polaire coördinaten

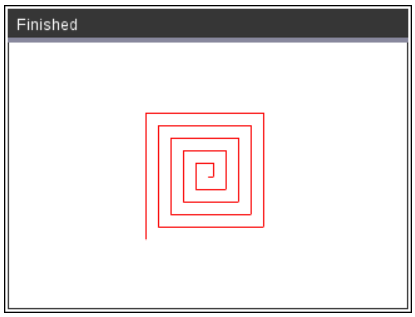
$$r(\theta) = 2a(1 - \cos(\theta))$$

Laat de Rover/Turtle een cardioïde rijden. Merk op dat de TI Rover-module een functies heeft om naar een punt gedefinieerd door poolcoördinaten te rijden: rv.to_polar(radius,theta).

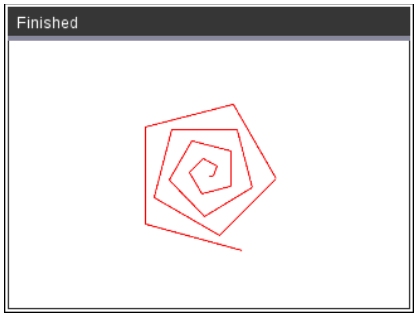
Opdracht 4: Spiraal-plezier

Gebruikmakend van de Turtle-module kan je met de volgende code creatief spiralen tekenen.

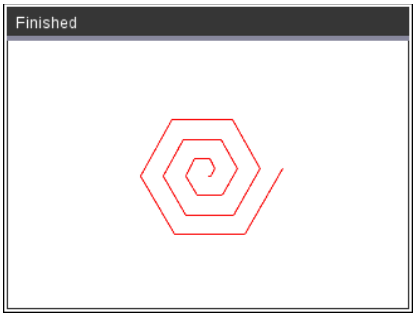
```
from turtle import *
rv=Turtle(); rv.hideturtle(); rv.hidegrid()

width=5
angle=90
segments=20

rv.pencolor(250,0,0)

for x in range(1,segments+1):
    rv.forward(x*width)
    rv.left(angle)
```



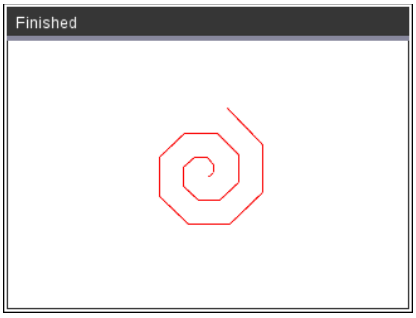
width=4 ; angle=75



width=3 ; angle=60

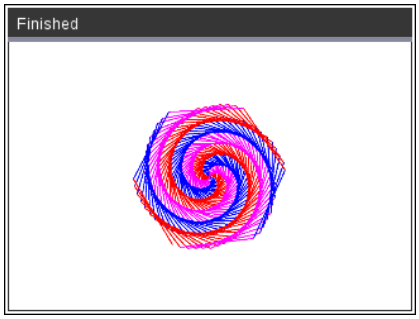


width=2 ; angle=45

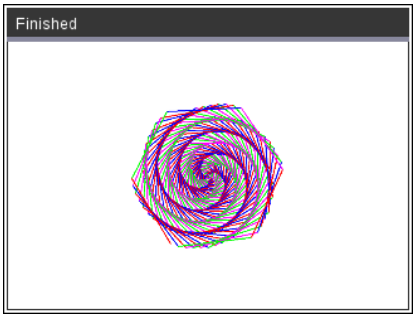


Experimenteren met kleur kan als volgt:

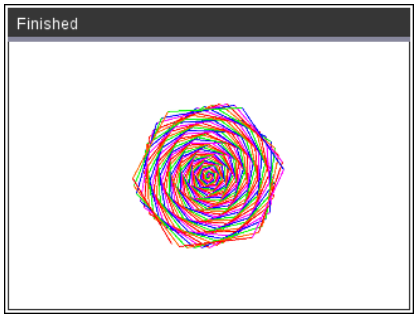
```
from turtle import *
rv=Turtle(); rv.hideturtle(); rv.hidegrid()

width=6
angle=59
segments=360

colors=['red','magenta','blue','green','orange','cyan']

for x in range(1,segments+1):
    rv.pencolor(colors[x%3])
    rv.forward(x/width)
    rv.left(angle)
```



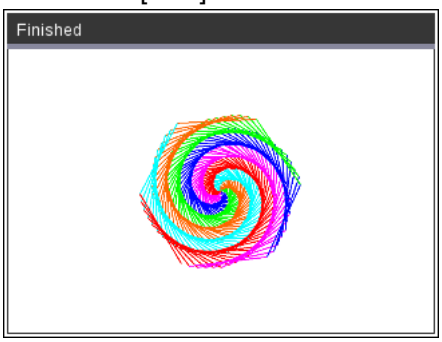
colors[x%4]



colors[x%5]

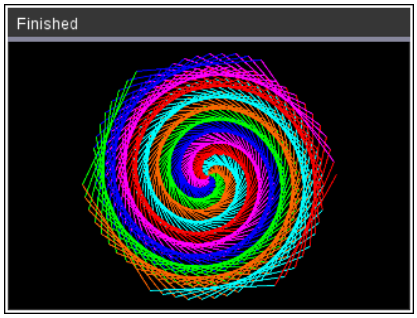


colors[x%6]

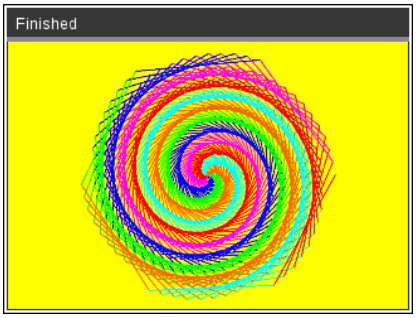


Een manier om met de Turtle-module, de achtergrond te kleuren is gebruik te maken van de dot()-functie(methode) en zo b.v. een gevulde cirkel tekenen met middelpunt de oorsprong en straal 200:

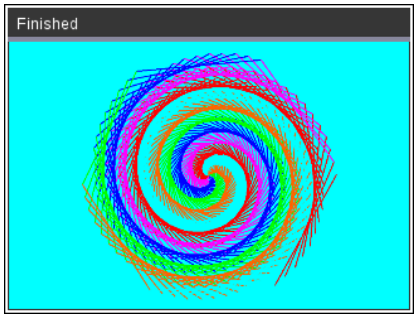
```
rv.pencolor('black')
rv.dot(200)
```



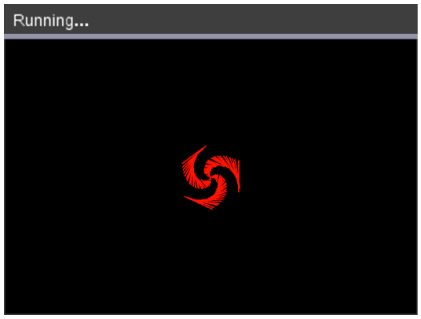

```
rv.pencolor('yellow')
rv.dot(200)
```

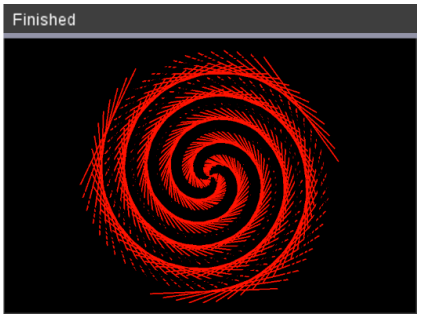



```
rv.pencolor('cyan')
rv.dot(200)
```



Veel plezier met het kunstig tekenen van spiralen!





Verdieping

a. Dashboard

Bij het lezen/meten van sensoren kan met de statements text_at() en cls() van de TI Hub-module een elementair dashboard gecodeerd worden in het grafisch (handheld) venster van de Python shell:

- text_at(row,"text","align") row = 1..13 en align: left – center – right
- cls() wissen van het output-venster

Voor het runnen van deze commando's moet het document in Handheld Page Size-mode staan.

Voor het meten van de ingebouwde lichthelderheid sensor kan dit als volgt:

```
from ti_hub import *
while get_key() != "esc":
    helderheid=brightness.measurement()
    cls()
    text_at(5,"De helderheid is","center")
    text_at(6,"{0:5.3f}".format(helderheid),"center")
    sleep(0.5)
    get_key()
```

De boodschap die in het dashboard wordt weergegeven moet als een string ingegeven worden. Met de stringmethode format() kunnen meetwaarden toegevoegd worden.

Voor de bovenstaande code flikkert het dashboard bij een refresh van het venster. Gebruik om dit te voorkomen de functies use_buffer() en paint_buffer() van de TI Draw-module (waarover meer in BootCamp 4).

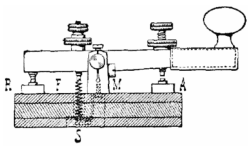
Het statement use_buffer() zorgt dat alle output voor het grafische venster van de Python shell uitgevoerd wordt in de achtergrond (geheugen) en dit tot paint_buffer() wordt uitgevoerd, dat alles van de buffer weergeeft in het venster.

```
from ti_hub import *
from ti_draw import *
use_buffer)
while get_key() != "esc":
    helderheid=brightness.measurement()
    cls()
    text_at(5,"De helderheid is","center")
    text_at(6,"{0:5.3f}".format(helderheid),"center")
    paint_buffer()
    sleep(0.5)
    get_key()
```

b. Morse-code

Morse-code werd in 1835 uitgevonden door Samuel Morse voor het versturen van berichten. Morse bestaat uit met tussenpauzes uitgezonden signalen. Voor de telegraaf had je de volgende keuzes uit twee toestanden:

- sleutel naar beneden (stroom) of naar boven (geen stroom)
- tijdsduur kort of lang

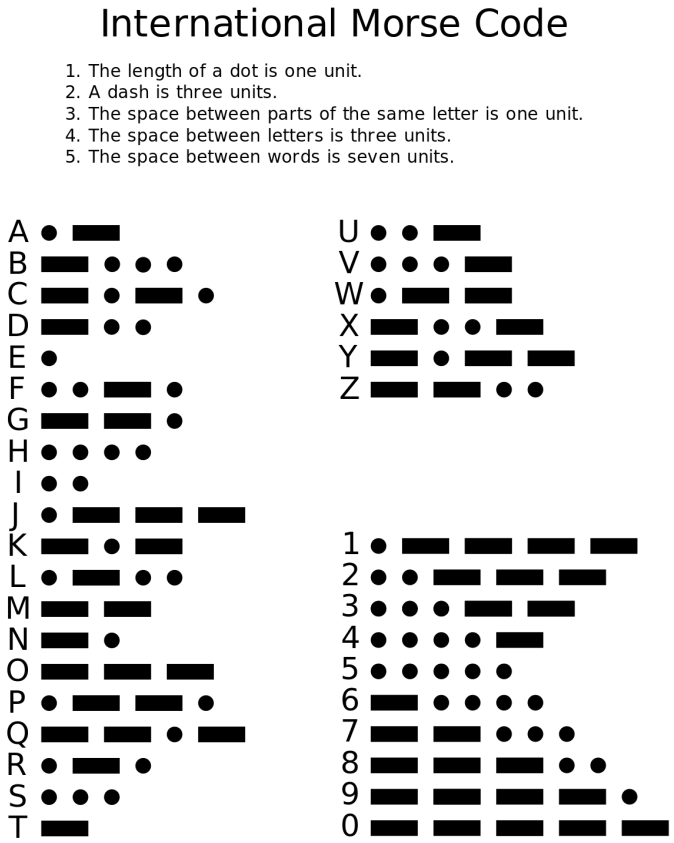


De hedendaagse Internationale Morse-code kent twee symbolen: punten en streepjes, ofwel *dits* en *dahs*. De lengte van de 'dit' bepaalt de snelheid waarmee de boodschap wordt verzonden en wordt als 'eenheid' gebruikt.

We coderen de Morse Code met als output een combinatie van de ingebouwde luidspreker en RGB-led en definiëren/coderen de Morse-code als een dictionary in een apart python programma alfabet.py.

alfabet.py

```
alfabet={
    "A": "·-",
    "B": "-···",
    "C": "-·-·",
    "D": "-··",
    "E": "·",
    "F": "··-·",
    "G": "--·",
    "H": "····",
    "I": "··",
    "J": "·---",
    "K": "-·-",
    "L": "·-··",
    "M": "--",
    "N": "-·",
    "O": "---",
    "P": "·--·",
    "Q": "--·-",
    "R": "·-·",
    "S": "···",
    "T": "-",
    "U": "··-",
    "V": "···-",
    "W": "·--",
    "X": "-··-",
    "Y": "-·--",
    "Z":"--··"
 }
```



Hieronder Python-code voor de TI-Innovator hub die Morse-code simuleert.

morse.py

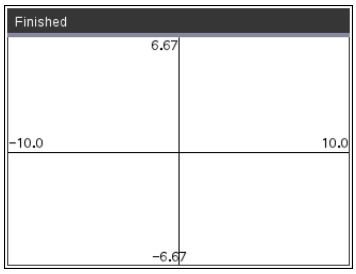
```
from ti_hub import *
from alfabet import *

#  Lengte kort signaal 0.11 s
t=0.11

#  Input boodschap en omzetten naar
 hoofdletters
while True:
tekst=input("Tekst (enter=stoppen): ")
    if tekst=="" or tekst[:3]=="esc":
        break
    tekst=tekst.upper()

#  Output in 650 Hz en witte flits
 rgb(255,255,255)
#     . = t en _ = 3*t
#     tijd tussen twee letters 3*t
#     tijd tussen twee worden 7*t
    for k in tekst:
        if k==" ":
            sleep(7*t)
        else:
            if k in alfabet:
                for d in alfabet[k]:
                    if d=="·":
                        x=t
                    else:
                        x=3*t
                    sound.tone(650,x)
                    color.rgb(255,255,255)
                    sleep(x+t)
                    color.off()
                sleep(3*t)
```

11. Setup Plot-omgeving

De TI PlotLib-module laat toe om grafieken en data-plots te tekenen. Voor deze grafische module moet voor TI-Nspire CX-software de document Preview-mode ingesteld zijn als Handheld.

We starten met het definiëren van een TI PlotLib-object, plt. Het hierna uitvoeren van de functie axes() toont de default instellingen van het plot-venster.

```
import ti_plotLib as plt
plt.axes(”on”)
```

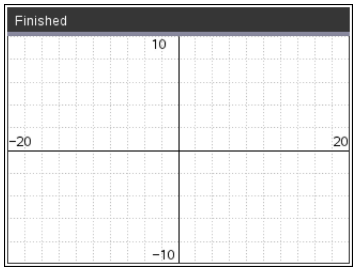


De volgende opties zijn voor axes() beschikbaar:

- “axes” alleen de assen, geen eindwaarden
- “window” alleen de eindwaarden, geen assen
- “off” geen assen en geen eindwaarden

Manueel instellen van het venster doe je met window(xmin,xmax,ymax,ymin) en met de functie grid(xScale,yScale,”style”) wordt een grid getekend:

```
import ti_plotLib as plt
plt.window(-20,20,-10,10)
plt.grid(1,1,”dotted”)
plt.axes(”on”)
```

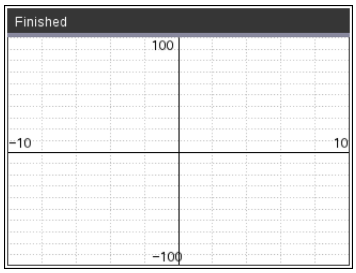


De opties voor de grid-stijl zijn:

“solid” _____ “dotted” ..... “dashed” - - - - -

Deze plt-vensterinstellingen kunnen ook gedefineerd worden als een functie:

```
import ti_plotLib as plt
def venster(xmin,xmax,ymin,ymax,xscale,yscale):
    plt.window(xmin,xmax,ymin,ymax)
    plt.grid(xscale,yscale,"dotted")
    plt.axes("on")
venster(-10,10,-100,100,2,10)
```

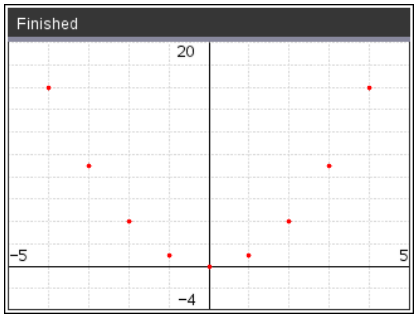


12. Grafieken

Het plotten van een grafiek kan beschouwd worden als het kleuren van de pixels van het scherm, t.o.v. een referentie-assenstelsel, die voldoen aan het verband voorgeschreven door de functiedefinitie.

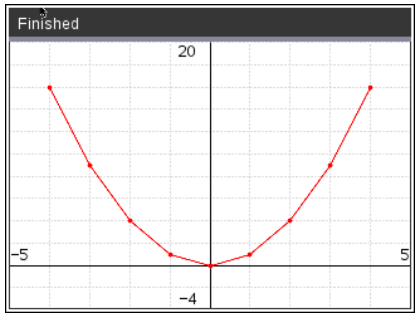
Een voorbeeld voor $f(x) = x^2$.

```
import ti_plotlib as plt
def venster(xmin,xmax,ymin,ymax,xscale,yscale):
    plt.window(xmin,xmax,ymin,ymax)
    plt.grid(xscale,yscale,"dotted")
    plt.axes("on")
def f(x):
    return x**2
xmin=-5 ; xmax=5 ; ymin=-4 ; ymax=20
venster(xmin,xmax,ymin,ymax,1,2)
for x in range(xmin+1,xmax):
    plt.color(255,0,0)
    plt.plot(x,f(x),"o")
```



Toevoegen van de onderstaande code verbindt de punten van de grafiek door een lijnstuk:

```
    if x < xmax-1:
        plt.line(x,f(x),x+1,f(x+1))
```

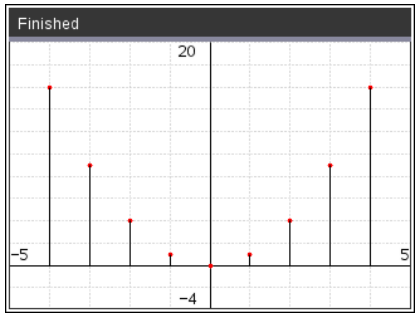


of de volgende code om de hiernaast afgebeelde 3e plot te genereren:

```
    plt.color(0,0,0)
    plt.pen("thin","solid")
    plt.line(x,f(x),x,0)
```

Voor de functie plot(x,y,"mark") zijn de volgende mark-opties beschikbaar:

- "o"
- "+"
- "x"
- "."



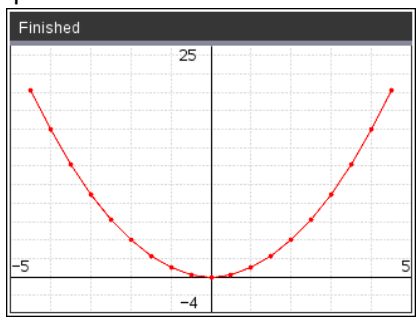
En voor pen("size","style"):

Size	Style
"thin"	"solid"
"medium"	"dotted"
"thick"	"dashed"

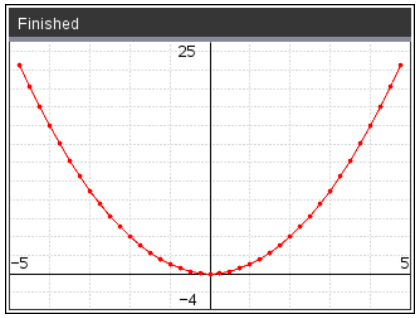
Het verfijnen van de te plotten grafiek kan als volgt:

```
import ti_plotlib as plt

def venster(xmin,xmax,ymin,ymax,xscale,yscale):
    plt.window(xmin,xmax,ymin,ymax)
    plt.grid(xscale,yscale,"dotted")
    plt.axes("on")

def f(x):
    return x**2

xmin=-5 ; xmax=5 ; ymin=-4 ; ymax=25
venster(xmin,xmax,ymin,ymax,1,2)

pix=20
res=(xmax-xmin)/pix
xrange=[xmin+n*res for n in range(1,pix)]

for x in xrange:
    plt.color(255,0,0)
    plt.plot(x,f(x),"o")
    if x < xmax-res:
        plt.line(x,f(x),x+res,f(x+res))
```

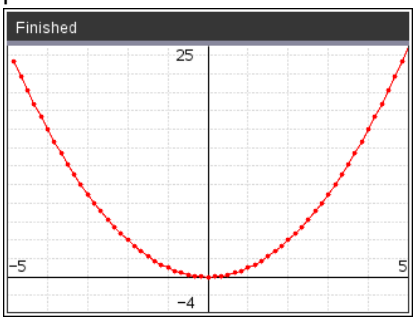
pix=20



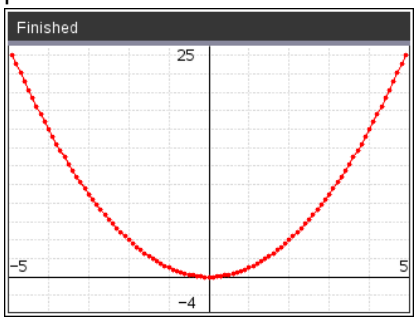
pix=40

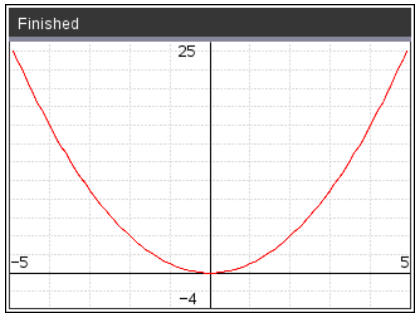


pix=60



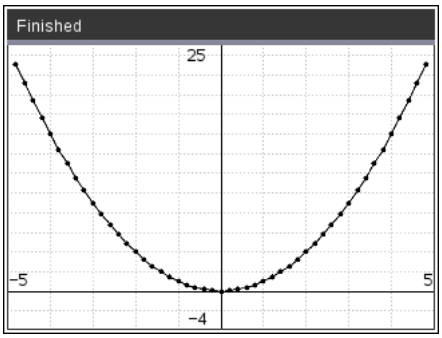
pix=100





I.p.v. voor plot() pixels te gebruiken als argumenten (en eventueel de pixels te verbinden met lijnstukjes), kan plot() ook gebruik maken van twee lijsten als argumenten en dit geeft een connected pixel plot als output.

```
import ti_plotlib as plt

def venster(xmin,xmax,ymin,ymax,xscale,yscale):
    plt.window(xmin,xmax,ymin,ymax)
    plt.grid(xscale,yscale,"dotted")
    plt.axes("on")

def f(x):
    return x**2

xmin=-5 ; xmax=5 ; ymin=-4 ; ymax=25
venster(xmin,xmax,ymin,ymax,1,2)

pix=50
res=(xmax-xmin)/pix
xrange=[xmin+n*res for n in range(1,pix)]
yrange=[f(i) for i in xrange]
plt.plot(xrange,yrange,"o")
```



13. Plotten van data

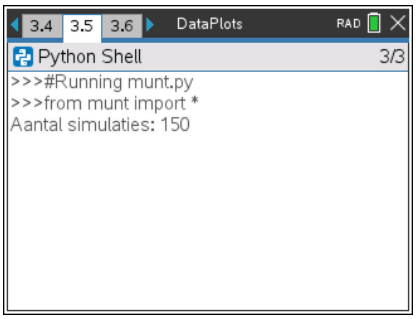
13.1. Kans-simulatie

We simuleren het opwerpen van een muntstuk, een kansexperiment met een binomiale kansverdeling:

- Kans op geen kop: $\frac{1}{4}$
- Kans op één keer kop: $\frac{1}{2}$
- Kans op twee keren kop: $\frac{1}{4}$

Voor het simuleren en herhaaldelijk uitvoeren van het experiment, runnen we de volgende code.

```
from random import *
import ti_plotlib as plt

def venster(xmin,xmax,ymin,ymax,xscale,yscale):
    plt.window(xmin,xmax,ymin,ymax)
    plt.axes("on")
```

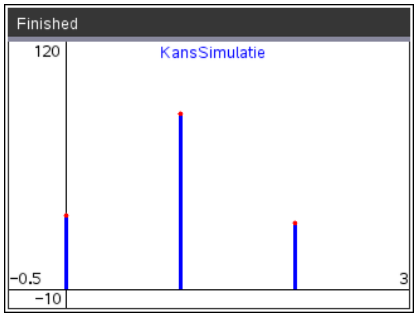


Definitie mogelijke uitkomsten en input aantal herhalingen experiment:

```
kop=[i for i in range(0,3)]
aantal=int(input("Aantal simulaties: "))
```

Herhaaldelijk uitvoeren experiment en opslaan resultaten:

```
for k in range(aantal):
    munt1=randint(0,1)
    munt2=randint(0,1)
    som=munt1+munt2
    kop[som]+=1
```

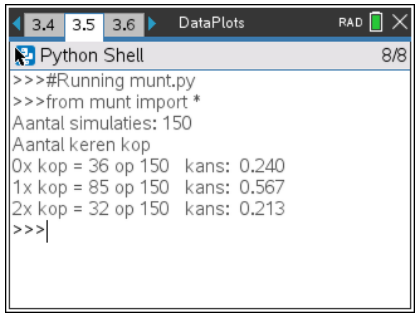


Definitie plot-venster en plotten van een titel:

```
xmin=-0.5 ; xmax=3 ; ymin=-10 ; ymax=int(aantal-20*aantal/100)
venster(xmin,xmax,ymin,ymax,1,10)

plt.color(0,0,255)
plt.title("KansSimulatie")
```

Plotten van de resultaten van de simulatie:

```
for x in range(0,3):
    plt.color(0,0,255)
    plt.pen("medium","solid")
    plt.line(x,0,x,kop[x])
    plt.color(255,0,0)
    plt.plot(x,kop[x],"o")
```

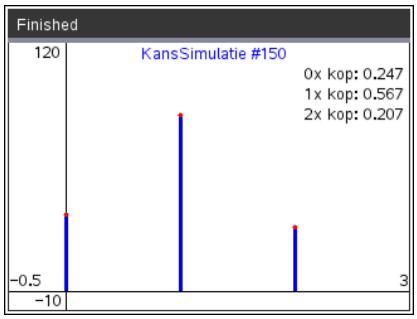


Weergeven van de benaderingen van de kans in de shell.

```
print("Aantal keren kop")
print("0x kop =",kop[0],"op",aantal,"  kans:  {0:2.3f}".format(kop[0]/aantal))
print("1x kop =",kop[1],"op",aantal,"  kans:  {0:2.3f}".format(kop[1]/aantal))
print("2x kop =",kop[2],"op",aantal,"  kans:  {0:2.3f}".format(kop[2]/aantal))
```

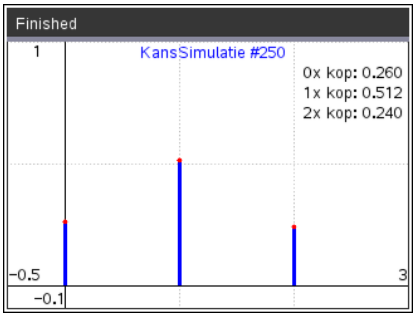
De kansbenaderingen kunnen ook als volgt weergegeven worden in het plot-venster:

```
plt.color(0,0,255)
plt.title("KansSimulatie #{}".format(aantal))
 ⋮
plt.color(0,0,0)
plt.text_at(2,"0x kop: {0:2.3f} ".format(kop[0]/aantal),"right")
plt.text_at(3,"1x kop: {0:2.3f} ".format(kop[1]/aantal),"right")
plt.text_at(4,"2x kop: {0:2.3f} ".format(kop[2]/aantal),"right")
```



In plaats van de resultaten kunnen ook door een kleine aanpassing van de code de kansbenaderingen geplot worden.

```
xmin=-0.5 ; xmax=3 ; ymin=-0.1 ; ymax=1
venster(xmin,xmax,ymin,ymax,1,0.5)
for x in range(0,3):
    plt.color(0,0,255)
    plt.pen("medium","solid")
    plt.line(x,0,x,kop[x]/aantal)
    plt.color(255,0,0)
    plt.plot(x,kop[x]/aantal,"o")
```

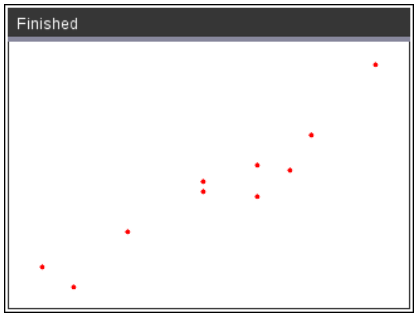


13.2. Puntenwolk

De methode(functie) scatter() plot een puntenwolk van twee datasets (lijsten).
We illustreren dit aan de hand van de lengte en gewicht van tien 18-jarigen:

Lengte	163	185	180	175	168	175	191	180	160	183
Gewicht	60	90	78	81	71	79	104	84	64	83

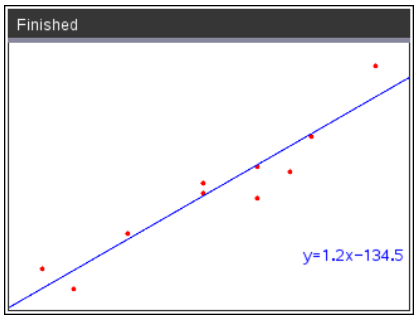
```
import ti_plotlib as plt
lengte=[163,185,180,175,168,175,191,180,160,183]
gewicht=[60,90,78,81,71,79,104,84,64,83]
plt.auto_window(lengte,gewicht)
plt.color(255,0,0)
plt.scatter(lengte,gewicht,"o")
```



Merk op dat in bovenstaande code auto_window() een venster creëert dat alle data bevat; di i.p.v. manueel een venster in te stellen baserend op de data.

De functie lin_reg() plot en berekent de beste lineaire benadering van de puntenwolk.

```
plt.color(0,0,255)
plt.lin_reg(lengte,gewicht,"right")
```



plt.m en plt.b geven de coëficiënten van de regressierechte:

```
Python Shell
>>>plt.m
1.2152619589977 22
>>>plt.b
-134.4861047835991
```

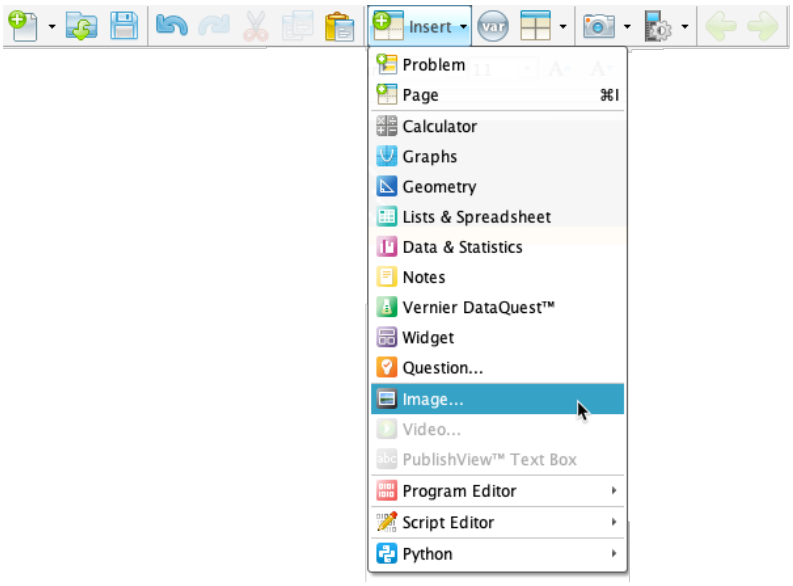
14. Digitale afbeeldingen

Digitale afbeeldingen kunnen beschouwd worden als matrices met als elementen pixel-waarden.

Onderstaande afbeelding van dimensie 769 x 505 wordt gerepresenteerd door een matrix van 388245 RGB-pixelwaarden als elementen. De kleur van ieder element/pixels is een 3-byte tuple.

Voor de onderstaande RGB 8-bits/channel afbeelding is dit 769 x 505 x 3 = 1165035 bytes.

Om een afbeelding te bewerken met Python voor TI-Nspire CX-technologie (de TI Image-module is niet beschikbaar voor de TI-84 Plus CE-T Python Edition) voegen we eerst een afbeelding toe aan een Notes-pagina. Klik op de afbeelding om ze te selecteren en geef de afbeelding een naam via een rechts-klik; b.v. ballon.

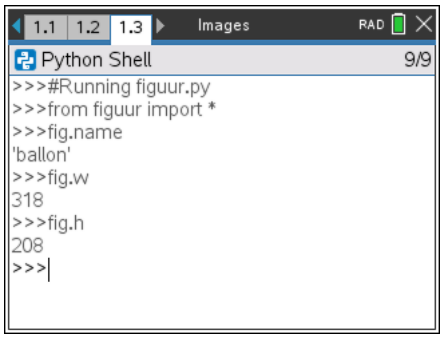


Het oproepen van de figuur ballon in een Python-programma gebeurt met deze code:


```
from ti_image import *
fig=load_image("ballon")
fig.show_image(0,0)
```

Voor een image-object zijn de volgende methodes beschikbaar:

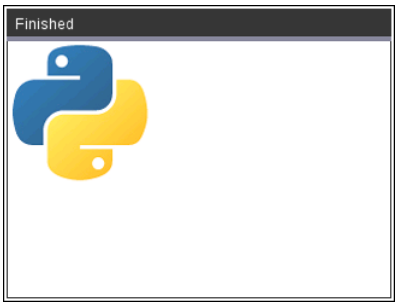
- get_pixel() leest de rgb-waarde van de pixel op posite (x,y),
- set_pixel(x,y,rgb-tuple) geeft de pixel op positie (x,y) de rgb-waarde bepaat door de tuple,
- show_image(x,y) toont de figuur met de linkerbovenhoek op positie (x,y),
- w, h, en name geeft respectievelijk de breedte, de hoogte of naam.

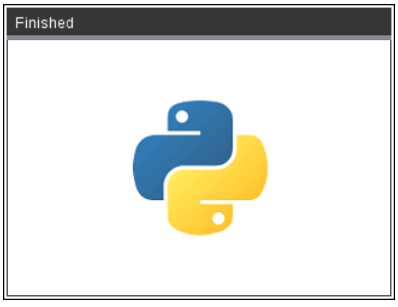


Merk op dat de dimensie van de figuur (maximaal) wordt aangepast i.f.v. de dimensie van het scherm; met behoud van de aspect ratio. De dimensie van het scherm is x: 0 ... 318 en y: 0 ... 212.

In de volgende voorbeelden van beeldverwerking, gebruiken we steeds show_image(0,0) voor het tonen van de afbeelding. Het centreren van de figuur in het grafische venster van de shell kan met:

$$\text{show\_image}\left(\frac{318-pyth.w}{2}, \frac{212-pyth.h}{2}\right)$$

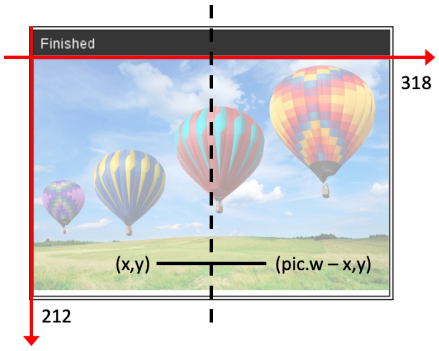


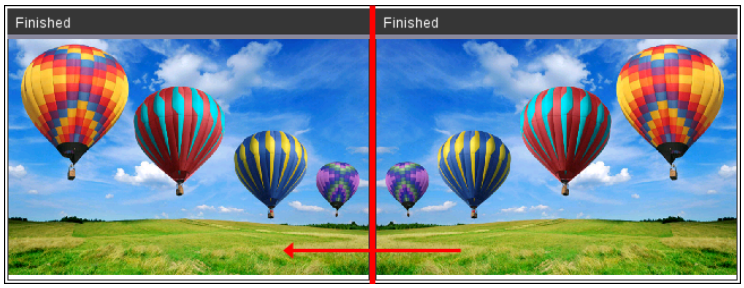


15. Transformaties

15.1. Flip Horizontaal

Voor het horizontaal flippen van de figuur “ballon” gebruiken we de volgende code:






```
from ti_image import *
pic=load_image("ballon")
newpic=copy_image(pic)
newpic.show_image(0,0)
for x in range(0,pic.w):
    for y in range(0,pic.h):
        rgb=pic.get_pixel((pic.w-1)-x,y)
        newpic.set_pixel(x,y,rgb)
    newpic.show_image(0,0)
```

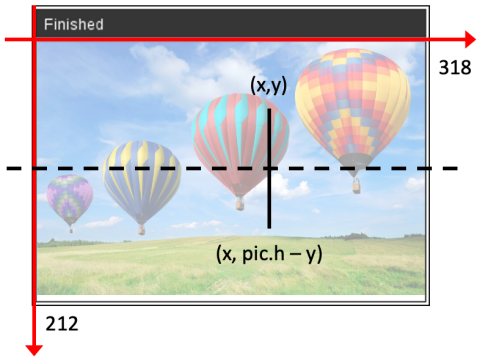
Merk op dat in de code gebruik gemaakt wordt van (pic.w – 1).

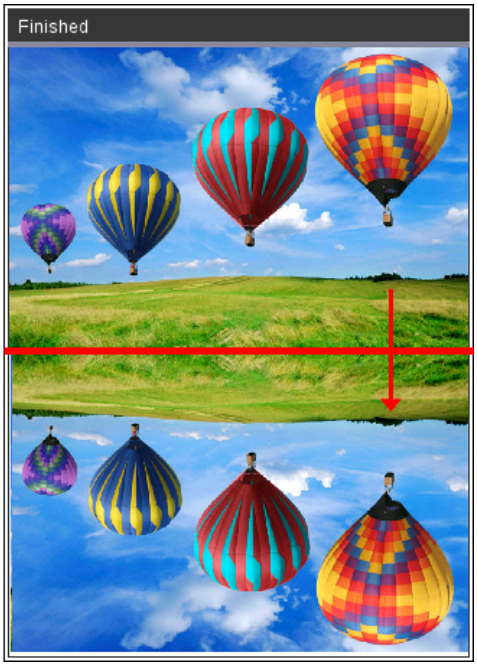
Voor een breedte pic.w (= 318) worden de pixels genummerd van 0 , ... , pic.w – 1 (=317).

Daar het statement newpic.show_image(0,0) behoort tot de for-lus voor x wordt de nieuwe figuur kolom per kolom gegeneerd.

Indien we dit statement uit de lus halen, wordt alles eerst berekent en dan getoond.

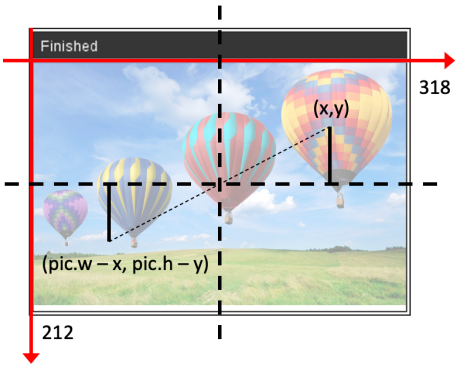
15.2. Flip Verticaal





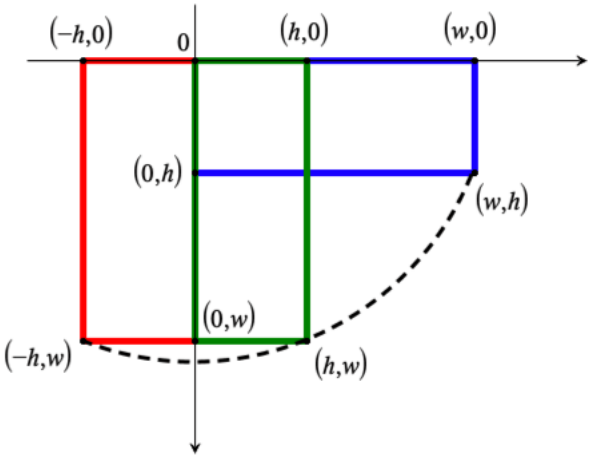

```
from ti_image import *
pic=load_image("ballon")
newpic=copy_image(pic)
newpic.show_image(0,0)
for x in range(0,pic.w):
    for y in range(0,pic.h):
        rgb=pic.get_pixel(x,(pic.h-1)-y)
        newpic.set_pixel(x,y,rgb)
    newpic.show_image(0,0)
```

15.3. Symmetrie




```
from ti_image import *
pic=load_image("ballon")
newpic=copy_image(pic)
newpic.show_image(0,0)
for x in range(0,pic.w):
    for y in range(0,pic.h):
        rgb=pic.get_pixel((pic.w-1)-x,(pic.h-1)-y)
        newpic.set_pixel(x,y,rgb)
    newpic.show_image(0,0)
```

15.4. Roteer 90° Rechts



Een rotatie van 90° naar rechts in het Python grafische venster maakt gebruik van de functie:

$$R: (x, y) \mapsto (-y, x)$$

Deze rotatie transformeert de blauwe rechthoek (pic) in de rode. Om de transformatie in het venster te krijgen, verschuiven we de rode rechthoek over een afstand pic.h; wat in de groene rechthoek resulteert (newpic):

$$G: (x, y) \mapsto (h - y, x)$$

Voor het programmeren van deze transformatie stellen we ons voor ieder pixel (p, q) van het beeld de volgende vraag. Welk pixel (x, y) van de originele afbeelding wordt afgebeeld op (p, q) m.a.w. $G(x, y) = (p, q)$:

$$G(x,y) = (p,q) \Leftrightarrow (h-y,x) = (p,q) \Leftrightarrow \begin{cases} h-y=p \\ x=q \end{cases} \Leftrightarrow \begin{cases} x=q \\ y=h-p \end{cases}$$

Indien de breedte van de afbeelding groter is dan de hoogte van het grafische venster (212 pixels) valt het beeld van de rotatie buiten dit venster. Vandaar passen we eerst een schaalverkleining toe i.f.v. de hoogte van het grafische venster. We gebruiken het statement clear() van de TI Draw-module om het scherm te wissen.

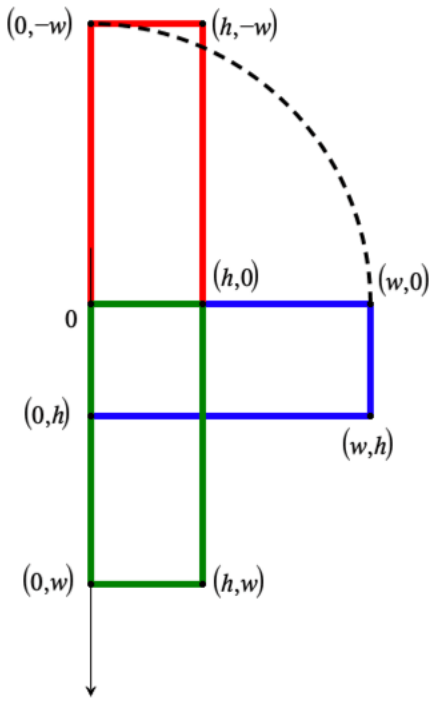
```
from ti_image import *
from ti_draw import *

pic=load_image("ballon")
pic.show_image(0,0)
```


```
xscale=int(212/318*pic.w)
yscale=int(212/318*pic.h)

npic=new_image(xscale,yscale,(255,255,255))

for x in range (0,npic.w):
    for y in range (0,npic.h):
        xp=int(318/212*x) ; yp=int(318/212*y)
        rgb=pic.get_pixel(xp,yp)
        npic.set_pixel(x,y,rgb)

clear()
npic.show_image(0,0)
```


```
newpic=new_image(npic.h,npic.w,(255,255,255))

clear()

for p in range (0,newpic.w):
    for q in range (0,newpic.h):
        rgb=npic.get_pixel(q,(npic.h-1)-p)
        newpic.set_pixel(p,q,rgb)
        newpic.show_image(0,0)
```

15.5. Roteer 90° Links

Voor het coderen van een rotatie van 90° naar links krijgen we een gelijkaardig verhaal.



Het uitvoeren van een rotatie van 90° naar links voor het Python grafische venster maakt gebruik van de functie:

$$R: (x, y) \mapsto (y, -x)$$

Deze rotatie transformeert de blauwe rechthoek (pic) in de rode. Om de transformatie in het venster te krijgen, verschuiven we de rode rechthoek over een afstand pic.w, wat in de groene rechthoek resulteert (newpic):

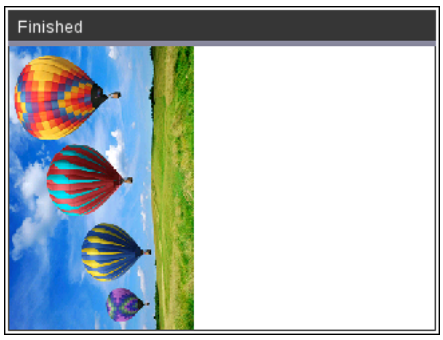
$$G: (x, y) \mapsto (y, w - x)$$

Voor het programmeren van deze transformatie stellen we ons voor ieder pixel (p, q) van het beeld de volgende vraag. Welk pixel (x, y) van de originele afbeelding wordt op afgebeeld op (p, q) m.a.w. $G(x, y) = (p, q)$:

$$G(x, y) = (p, q) \Leftrightarrow (y, w - x) = (p, q) \Leftrightarrow \begin{cases} y = p \\ w - x = q \end{cases} \Leftrightarrow \begin{cases} x = w - q \\ y = p \end{cases}$$

We passen de code voor de rotatie naar rechte als volgt aan voor een rotatie naar links:

```
for p in range (0,newpic.w):
    for q in range (0,newpic.h):
        rgb=npic.get_pixel((npic.w-1)-q),p)
        newpic.set_pixel(p,q,rgb)
        newpic.show_image(0,0)
```



16. Filters

We bekijken enkele algoritmes i.v.m. image processing die de kleur, contrast en helderheid van een afbeelding/foto veranderen. Voor al deze algoritmes voeren we een transformatie uit op de rgb-waarde van ieder pixel.

16.1. Grayscale

Om een figuur te converteren naar grayscale kunnen we gebruik maken van de intensiteit of de helderheid van de kleur. Een manier om dat te doen is het berekenen van de gemiddelde waarde van rood, groen en blauw:

$$I = \frac{Rood + Groen + Blauw}{3}$$

Deze methode is verre van ideaal daar onze ogen de intensiviteit van rood, groen en blauw niet op dezelfde manier interpreteren. Bijvoorbeeld groen kleurt veel helderder bij maximale intensiteit dan blauw.

Om rekening te houden met onze perceptie van kleuren kennen we als volgt een gewicht toe aan iedere kleur om de intensiteit te bereken:

$$I = 0.299 \cdot Rood + 0.587 \cdot Groen + 0.114 \cdot Blauw$$

We bekijken even het verschil van beide filters voor de onderstaande kleuren:

```
from ti_image import *
pic=load_image("kleur")
pic.show_image(0,0)
```

Gemiddelde intensiteit

```
for x in range (0,pic.w):
    for y in range (0,pic.h):
        rgb = pic.get_pixel(x,y)
        sum = int((rgb[0] + rgb[1] + rgb[2]) /
3)
        gray = (sum,sum,sum)
        pic.set_pixel(x,y,gray)
pic.show_image(0,0)
```

Gewogen intensiteit

```
for x in range (0,pic.w):
    for y in range (0,pic.h):
        rgb = pic.get_pixel(x,y)
        r = rgb[0] * .299
        g = rgb[1] * .587
        b = rgb[2] * .114
        sum = int(r + g + b)
        gray = (sum,sum,sum)
        pic.set_pixel(x,y,gray)
pic.show_image(0,0)
```

Voor de afbeelding ballon geeft dit de volgende output:

Gemiddelde intensiteit

Gewogen intensiteit

16.2. Sepia

De Sepia-filter is een filter die o.a. in fotografie gebruik om een foto een rood-bruine kleur te geven.

Voor de Sepia-filter worden de volgende transformaties toegepast:

- newRood = int(0.393 · Rood + 0.769 · Groen + 0.189 · Blauw)
- newGroen = int(0.349 · Rood + 0.686 · Groen + 0.168 · Blauw)
- newBlauw = int(0.272 · Rood + 0.534 · Groen + 0.131 · Blauw)

Indien na toepassing van de filter een waarde groter is dan 255, declareer de waarde als 255.

```
from ti_image import *
pic=load_image("ballon")
pic.show_image(0,0)
for x in range(0,pic.w):
    for y in range(0,pic.h):
        rgb=pic.get_pixel(x,y)
        r=int(rgb[0]*0.393+rgb[1]*0.769+rgb[2]*0.189)
        if r > 255:
            r=255
        g=int(rgb[0]*0.349+rgb[1]*0.686+rgb[2]*0.168)
        if g > 255:
            g=255
        b=int(rgb[0]*0.272+rgb[1]*0.534+rgb[2]*0.131)
        if b > 255:
            b=255
        sepia=(r,g,b)
        pic.set_pixel(x,y,sepia)
    pic.show_image(0,0)
```

16.3. Inversie & Solarisering

Eén van de eenvoudigste vormen van image processing is inversie of het transformeren naar een negatief.

De filter die hier gebruikt wordt is:

- newRood = 255 – Rood
- newGroen = 255 – Groen
- newBlauw = 255 – Blauw

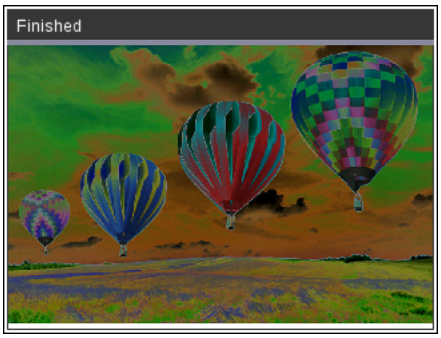
```
from ti_image import *
pic=load_image("ballon")
pic.show_image(0,0)
for x in range(0,pic.w):
    for y in range(0,pic.h):
        rgb=pic.get_pixel(x,y)
        r=255-rgb[0]
        g=255-rgb[1]
        b=255-rgb[2]
        inv=(r,g,b)
        pic.set_pixel(x,y,inv)
    pic.show_image(0,0)
```

Gelijkaardig aan de negatief-filter is de solaris-filter, het verschil is dat voor het solaris-effect de negatief-filter enkel wordt toegepast voor kleur-waarden groter (of kleiner) dan een bepaalde waarde, de treshold.

```
from ti_image import *
pic=load_image("ballon")
pic.show_image(0,0)
treshold=128
for x in range(0,pic.w):
    for y in range(0,pic.h):
        rgb=list(pic.get_pixel(x,y))
        for i in range(0,3):
            if rgb[i] < treshold:
                rgb[i]=255-rgb[i]
        srgb=(rgb[0],rgb[1],rgb[2])
        pic.set_pixel(x,y,srgb)
    pic.show_image(0,0)
```

rgb[i[] < treshold (=128)

rgb[i[] > treshold (=128)



16.4. Helderheid

Het veranderen van de helderheid van een afbeelding is best eenvoudig. Het komt neer op de gewenste verandering toe te voegen aan iedere rgb-waarde van een pixel.

Voor de waarde van verandering wordt meestal een integer gekozen tussen -225 en 255. Negatieve waarden maken de afbeelding donkerder en positieve waarden maken de afbeelding helderder.

Om te voorkomen dat een rgb-waarde groter dan 255 wordt of kleiner dan 0, passen we de volgende truncate-functie toe:

```
def truncate(value):
    if value < 0:
        value=0
    if value > 255:
        value=255
    return value
```

Voor het aanpassen van de helderheid gebruiken we de volgende code:

```
from ti_image import *
def truncate(value):
    if value < 0:
        value=0
    if value > 255:
        value=255
    return value
factor=int(input("Factor -255 < ... < 255: "))
pic=load_image("ballon")
pic.show_image(0,0)
for x in range (0,pic.w):
    for y in range (0,pic.h):
        rgb=pic.get_pixel(x,y)
        r=int(truncate(rgb[0]+factor))
        g=int(truncate(rgb[1]+factor))
        b=int(truncate(rgb[2]+factor))
        rgb=(r,g,b)
        pic.set_pixel(x,y,rgb)
    pic.show_image(0,0)
```

factor = 75

factor = -75

16.5. Contrast

Het veranderen van het contrast verloopt wat complexer dan het aanpassen van de helderheid:

a. Baserend of het ingegeven contrast-level, c, bepalen we de contrast-correctiefactor, F, als volgt:

$$F = \frac{259(c+255)}{255(259-c)}$$

b. Met deze correctiefactor voeren we de volgende pixel-transformatie uit:

- newRood = int(F · (Rood – 128) + 128))
- newGroen = int(F · (Groen – 128) + 128))
- newBlauw = int(F · (Blauw – 128) + 128))

```
from ti_image import *
def truncate(value):
    if value < 0:
        value=0
    if value > 255:
        value=255
    return value
contrast=int(input("Factor -255 < ... < 255: "))
pic=load_image("ballon")
pic.show_image(0,0)
for x in range (0,pic.w):
    for y in range (0,pic.h):
        rgb=pic.get_pixel(x,y)
        factor=(259*(contrast+255))/(255*(259-contrast))
        r=int(truncate(factor*(rgb[0]-128)+128))
        g=int(truncate(factor*(rgb[1]-128)+128))
        b=int(truncate(factor*(rgb[2]-128)+128))
        rgb=(r,g,b)
        pic.set_pixel(x,y,rgb)
    pic.show_image(0,0)
```

contrast = 75

contrast = -75

17. TI Draw Basics

De module TI Draw bevat een aantal functies(methodes) die het toelaat om segmenten, rechthoeken, veelhoeken, cirkels, bogen en tekst toe te voegen aan het grafische venster van de shell.

De default-configuratie heeft (0,0) als linkerbovenhoek met de positieve richting van de x-as naar rechts en de positieve richting van de y-as naar beneden.

```
from ti_draw import *
draw_text(2,16,"(0,0)")
draw_text(262,210,"(318,212)")
```

Deze configuratie is niet zo vertrouwelijk als de oriëntatie van een standaard wiskundig assenstelsel. Hiervoor veranderen we de oriëntatie van het y-as als volgt:

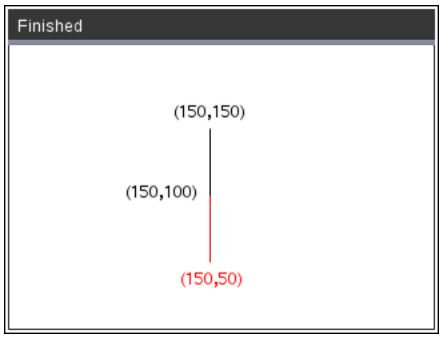
```
from ti_draw import *
dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])
draw_text(2,2,"(0,0)")
draw_text(262,195,"(318,212)")
```

We verduidelijken de beschikbare functies voor het tekenen van wiskundige figuren.

17.1. Segmenten

draw_line(x1,y1,x2,y2) tekent een segment tussen de punten (x1,y1) en (x2,y2).

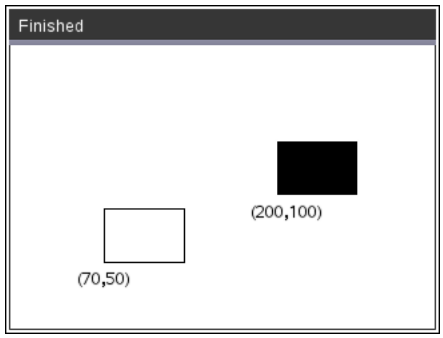
```
from ti_draw import *
dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])
draw_line(150,100,150,150)
draw_text(87,95,"(150,100)")
draw_text(123,155,"(150,150)")
set_color(255,0,0)
draw_line(150,100,150,50)
draw_text(128,30,"(150,50)")
```



17.2. Rechthoeken

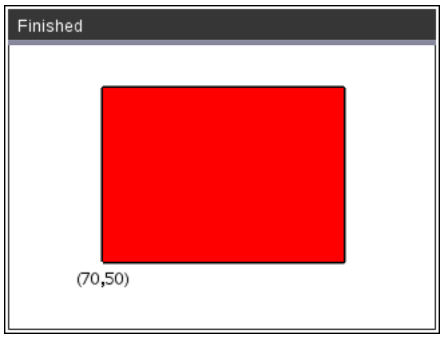
draw_rect(x,y,breedte,hoogte) – fill_rect(x,y,breedte,hoogte) tekent een rechthoek zoals hieronder aangegeven met (x,y) als linkerbenedenhoek.

```
from ti_draw import *
dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])
draw_rect(70,50,60,40)
draw_text(50,30,"(70,50)")
fill_rect(200,100,60,40)
draw_text(180,80,"(200,100)")
```



Een rechthoek kan ook getekend worden als veelhoek d.m.v lijsten met de coördinaten van de hoekpunten van de rechthoek.

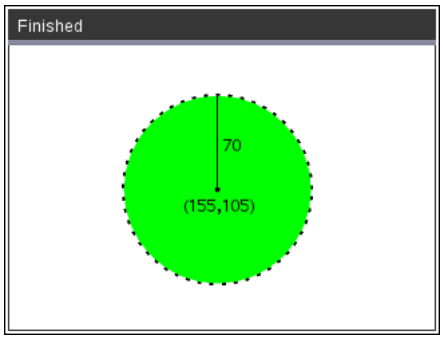
```
from ti_draw import *
dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])
draw_text(50,30,"(70,50)")
xcoord=[70,250,250,70,70]
ycoord=[50,50,180,180,50]
set_pen("medium","solid")
draw_poly(xcoord,ycoord)
set_color(255,0,0)
fill_poly(xcoord,ycoord)
```



17.3. Cirkels

draw_circle(x,y,straal) – fill_circle(x,y,straal) tekent een cirkel met middelpunt (x,y).

```
from ti_draw import *
dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])
set_pen("medium","dotted")
draw_circle(155,105,70)
set_color(0,255,0)
fill_circle(155,105,70)
set_color(0,0,0)
fill_circle(155,105,2)
draw_text(130,85,"(155,105)")
set_pen("thin","solid")
draw_text(160,130,"70")
draw_line(155,105,155,175)
```

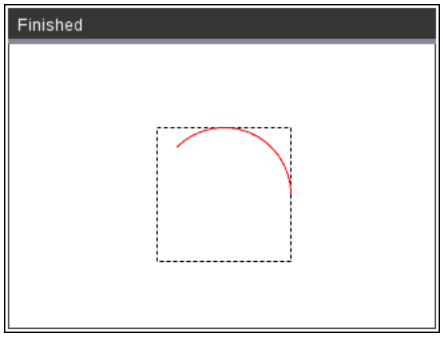


17.4. Cirkelbogen

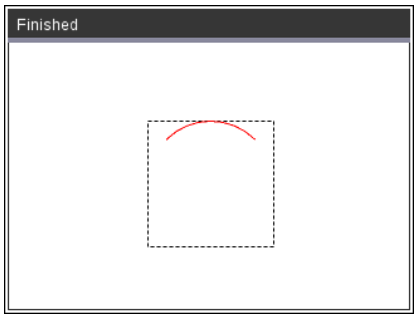
draw_arc(x,y,breedte,hoogte,beginhoek,hoekgrootte) –
fill_arc(x,y,breedte,hoogte,beginhoek,hoekgrootte)

Deze functie tekent een boog op de cirkel ingesloten in de rechthoek, draw_rect(x,y,breedte,hoogte), waarbij (x,y) het linkerbenedenhoekpunt is t.o.v. het door ons gedefinieerde assenstelsel/window.

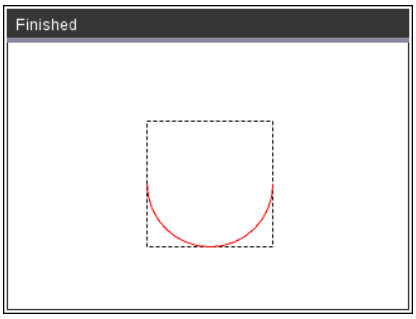
```
from ti_draw import *
dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])
h=100
set_pen("thin","dashed")
draw_rect(110,50,h,h)
set_color(255,0,0)
set_pen("thin","solid")
draw_arc(110,50,h,h,0,135)
```



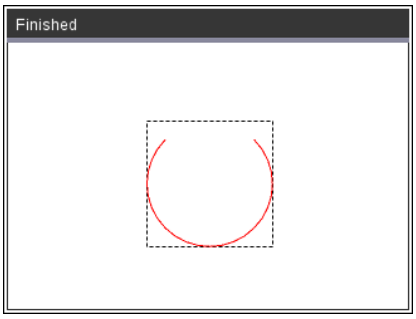
Nog enkele voorbeelden



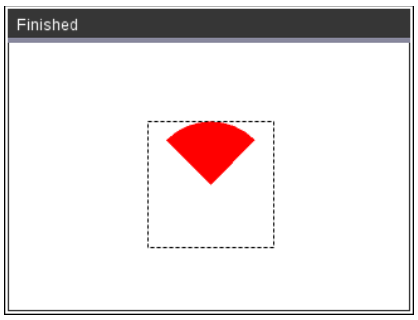
draw_arc(110,50,100,100,45,90)



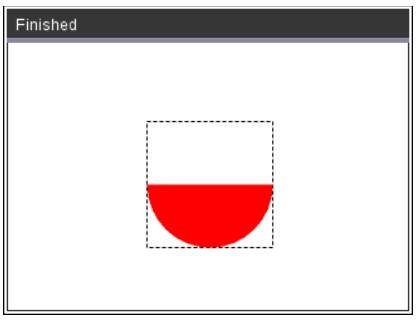
draw_arc(110,50,100,100,180,180)



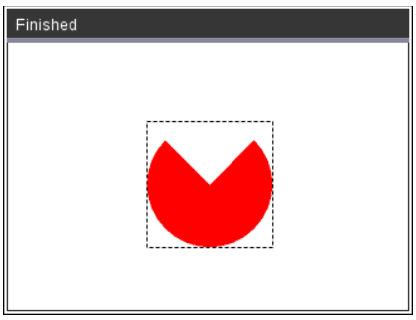
draw_arc(110,50,100,100,135,270)



fill_arc(110,50,100,100,45,90)



fill_arc(110,50,100,100,180,180)



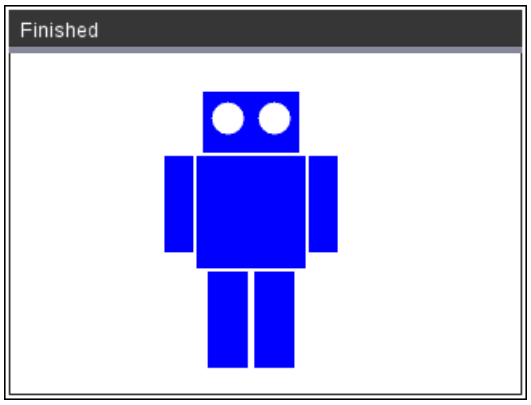
fill_arc(110,50,100,100,135,270)

18. Creatief met lijnen en bogen

18.1. Robot

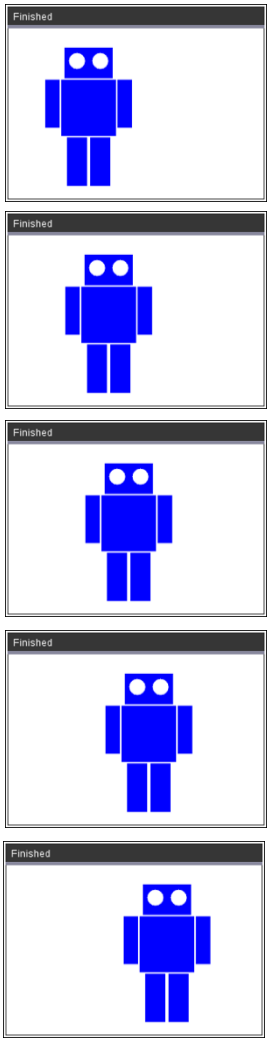
Met enkele rechthoeken en twee cirkels tekenen we onderstaande robot:

```
from ti_draw import *

dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])

set_color(0,0,255)
fill_rect(120,150,60,38)
fill_rect(116,78,68,70)
fill_rect(96,88,18,60)
fill_rect(186,88,18,60)
fill_rect(123,16,25,60)
fill_rect(152,16,25,60)
set_color(255,255,255)
fill_circle(135,172,10)
fill_circle(164,172,10)
```



Het tekenen van de robot in een for-loop plaatsen,
zet de robot in beweging:

```
from ti_draw import *

dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])

use_buffer()

for i in range(-50,50):
    clear()
    set_color(0,0,255)
    fill_rect(120+i,150,60,38)
    fill_rect(116+i,78,68,70)
    fill_rect(96+i,88,18,60)
    fill_rect(186+i,88,18,60)
    fill_rect(123+i,16,25,60)
    fill_rect(152+i,16,25,60)
    set_color(255,255,255)
    fill_circle(135+i,172,10)
    fill_circle(164+i,172,10)
    paint_buffer()
```



Zonder gebruik te maken van de buffer()-statements, genereert
de for-lus een flikkerend beeld van de bewegende robot.

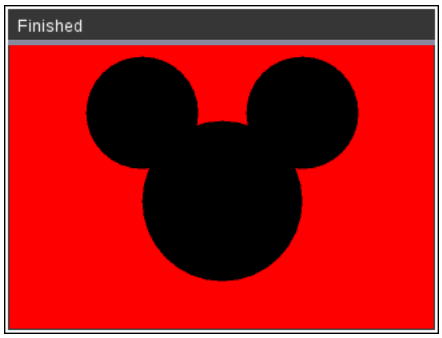
use_buffer() zorgtdat alles in de achtergrond(geheugen) wordt getekend totdat
paint_buffer() de buffer tekent.

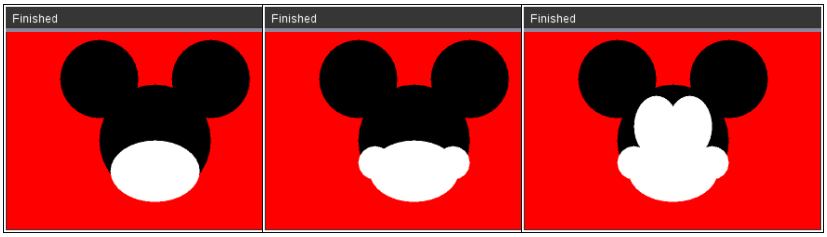
Het is aan te raden de buffer te gebruiken indien veel objecten getekend worden
(snelheid) en indien het scherm regelmatig wordt gewist voor nieuwe objecten
(flikkeren).

18.2. Mickey

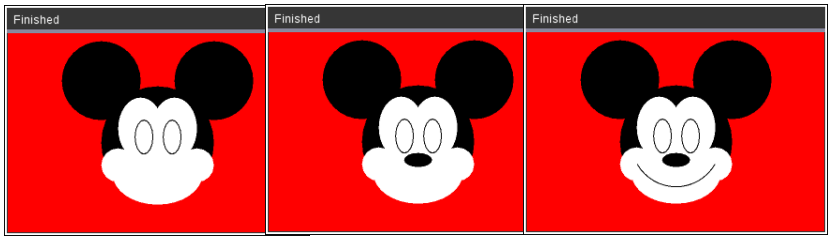
Voor het tekenen van Mickey configureren we het scherm met de oorsprong in het midden:

```
from ti_draw import *

dim=get_screen_dim()
set_window(-dim[0]/2,dim[0]/2,-dim[1]/2,dim[1]/2)

set_color(255,0,0)
fill_rect(-dim[0]/2,-dim[1]/2,318,212)

set_color(0,0,0)
fill_circle(0,-10,60)
fill_circle(-60,56,42)
fill_circle(60,56,42)

set_color(255,255,255)
fill_arc(-48,-76,96,66,0,360)

fill_circle(42,-34,18)
fill_circle(-42,-34,18)

fill_arc(-42,-28,48,66,0,360)
fill_arc(-6,-28,48,66,0,360)
```



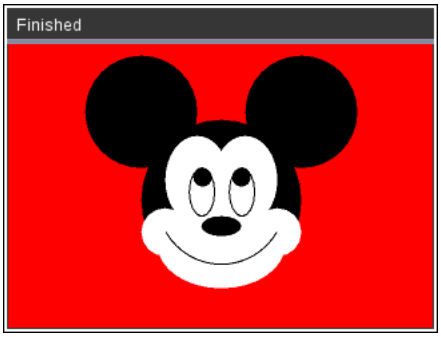


```
set_color(0,0,0)
draw_arc(-24,-22,19,36,0,360)
draw_arc(6,-22,19,36,0,360)

fill_arc(-15,-37,30,15,0,360)

draw_arc(-48,-58,96,96,210,120)
```



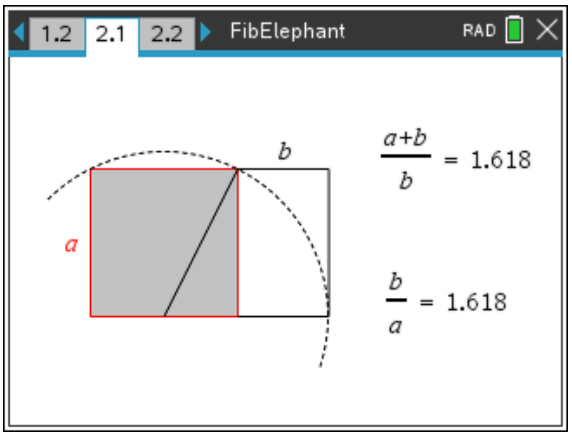
En tenslotte de pupillen van de ogen:

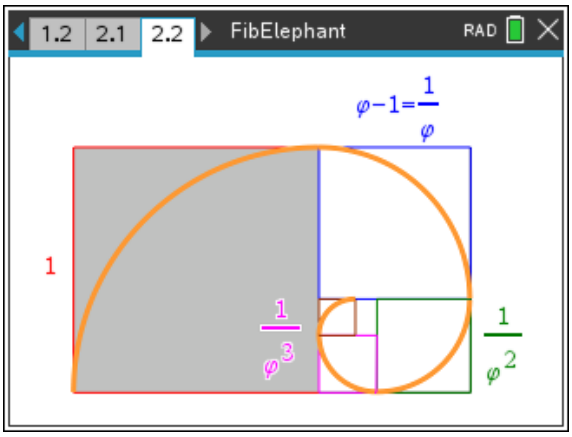
```
fill_circle(15,7.3,7)
fill_circle(-14,7.3, 7)
```



18.3. De olifant van Fibonacci

Een gouden rechthoek is een rechthoek waarvan de lengte en breedte zich verhouden als de gulden snede. $\varphi = \frac{1+\sqrt{2}}{5}$. Een gouden spiraal is een logaritmische spiraal die groeit met een factor φ.



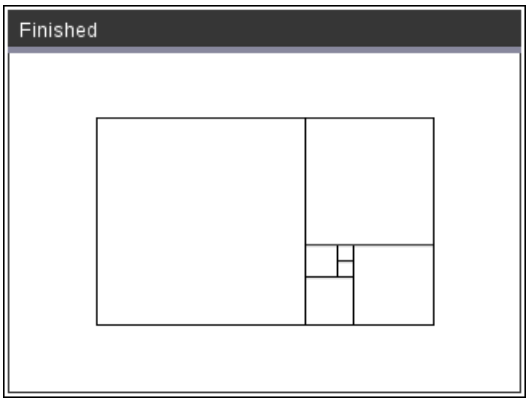


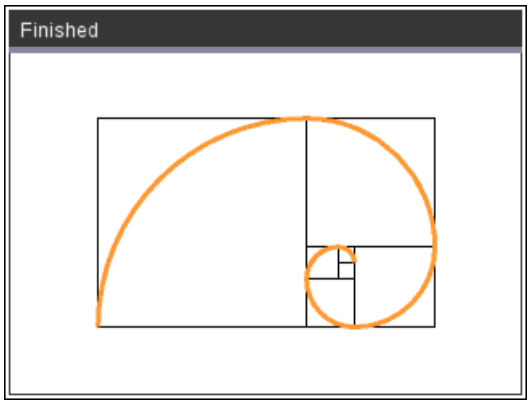
Vermits de rij met verhoudingen van opeenvolgende Fibonacci-getallen convergeert naar φ, kunnen we met de rij van Fibonacci een goudenspiraal benaderen.

De volgende code tekent de Fibonacci-spiraal:

```
from ti_draw import *

dim=get_screen_dim()
set_window(-54,dim[0]-54,-42,dim[1]-42)

draw_rect(0,0,130,130)
draw_rect(130,50,80,80)
draw_rect(160,0,50,50)
draw_rect(130,0,30,30)
draw_rect(130,30,20,20)
draw_rect(150,40,10,10)
```

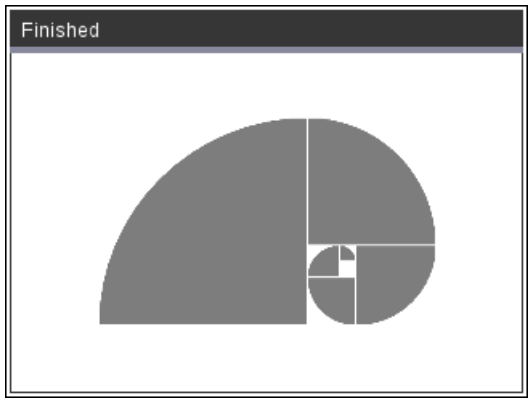
```
set_pen("medium","solid")
set_color(255,153,50)
z=130 ; draw_arc(0,0-z,2*z,2*z,90,90)
z=80 ; draw_arc(130-z,50-z,2*z,2*z,0,90)
z=50 ; draw_arc(160-z,0,2*z,2*z,270,90)
z=30 ; draw_arc(130,0,2*z,2*z,180,90)
z=20 ; draw_arc(130,30-z,2*z,2*z,90,90)
z=10 ; draw_arc(150-z,40-z,2*z,2*z,0,90)
```

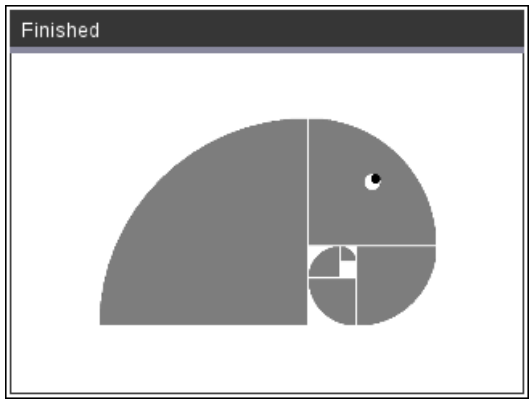




Het inkleuren van de cirkelsectoren en wat aanpassen van de kleuren en de volgorde van plotten, geeft de volgende figuur – de olifant van Fibonacci.

```
from ti_draw import *

dim=get_screen_dim()
set_window(-54,dim[0]-54,-42,dim[1]-42)

set_color(125,125,125)
z=130 ; fill_arc(0,0-z,2*z,2*z,90,90)
z=80 ; fill_arc(130-z,50-z,2*z,2*z,0,90)
z=50 ; fill_arc(160-z,0,2*z,2*z,270,90)
z=30 ; fill_arc(130,0,2*z,2*z,180,90)
z=20 ; fill_arc(130,30-z,2*z,2*z,90,90)
z=10 ; fill_arc(150-z,40-z,2*z,2*z,0,90)

set_color(255,255,255)
draw_rect(0,0,130,130)
draw_rect(130,50,80,80)
draw_rect(160,0,50,50)
draw_rect(130,0,30,30)
draw_rect(130,30,20,20)
draw_rect(150,40,10,10)

fill_circle(170,90,5)
set_color(0,0,0)
fill_circle(172,92,3)
```





19. Iteratieve grafische algoritmes

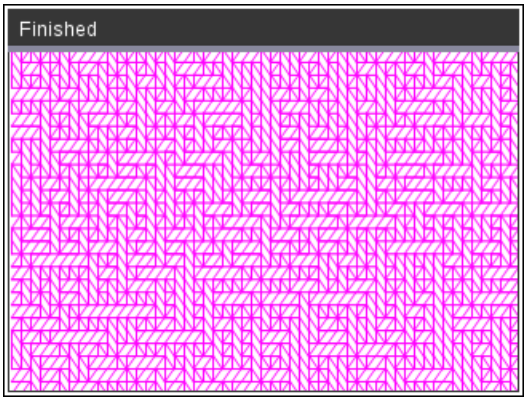
Voor de volgende voorbeelden gebruiken we de standaard vensterinstellingen.

19.1. At random creatief met tekst

Door de code at random te laten kiezen tussen twee karakters vullen we het grafische venster:

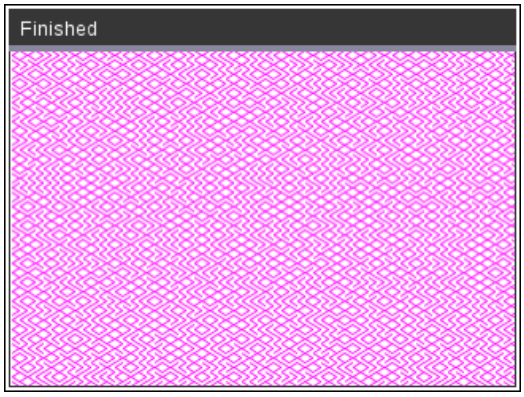
"N" of "Z"

```
from random import *
from ti_draw import *
use_buffer()
set_color(255,0,255)
for j in range(10,220,8):
    for i in range(0,318,6):
        if randint(0,1) > 0:
            draw_text(i,j,"N")
        else:
            draw_text(i,j,"Z")
    paint_buffer()
```



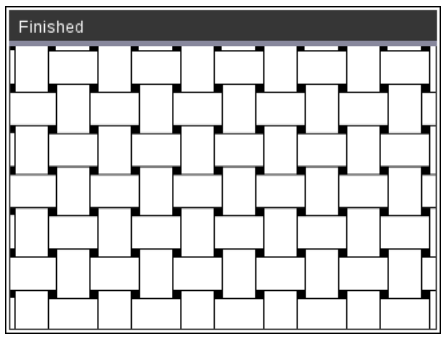
"<" of ">"

```
from random import *
from ti_draw import *
use_buffer()
set_color(255,0,255)
for j in range(10,220,6):
    for i in range(0,318,5):
        if randint(0,1) > 0:
            draw_text(i,j,"N")
        else:
            draw_text(i,j,"Z")
    paint_buffer()
```



19.2. Modulo-matje weven

Met de volgende code tekenen we deze figuur:



Hiervoor verdelen we het scherm horizontaal en verticaal in d (=31) delen met s (=3) spatie tussen de verticale en horizontale banden.

```
from ti_draw import *
use_buffer()
d=31
s=3
```

Hoe coderen we de verticale banden?

De volgende lus geeft:

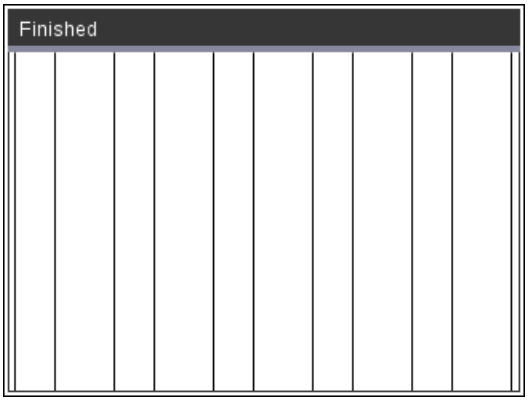
```
for i in range(0,318,d):
    x=i+s
    y=0
    draw_line(x,y,x,y+d)
paint_buffer()
```

We willen nu de x-coordinaat afwisselend vermeerderen met s en verminderen met s.
Dit kan als volgt m.b.v. modulo 2:

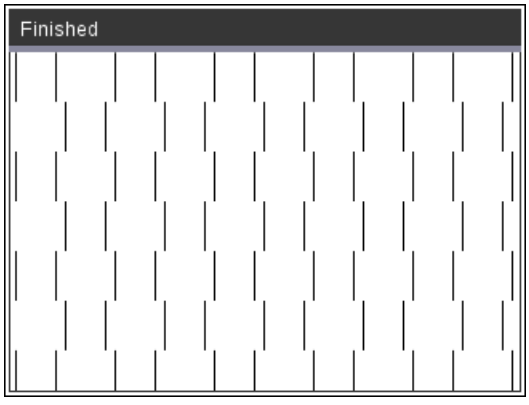
```
for i in range(0,318,d):
    if i%2 == 0:
        x=i+s
    else:
        x=i-s
    y=0
    draw_line(x,y,x,y+d)
paint_buffer()
```

Indien we dit verticaal herhalen met stap grootte d=31 geeft dit het linkse resultaat, niet wat we nodig hebben hier. Daarom voegen we de variabele j toe (rechts) aan de modulo-check:

```
for j in range(0,212,d):
    for i in range(0,318,d):
        if i%2 == 0:
            x=i+s
        else:
            x=i-s
        y=j
        draw_line(x,y,x,y+d)
paint_buffer()
```

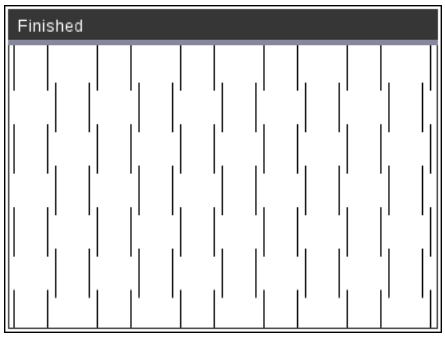



```
for j in range(0,212,d):
    for i in range(0,318,d):
        if (i+j)%2 == 0:
            x=i+s
        else:
            x=i-s
        y=j
        draw_line(x,y,x,y+d)
paint_buffer()
```



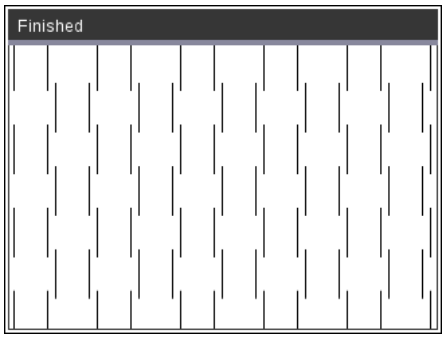
Als laatste stap verlengen we de lengte van de lijnstukkenmet 2*s, een afstand s omhoog en omlaag:

```
for j in range(0,212,d):
    for i in range(0,318,d):
        if (i+j)%2 == 0:
            x=i+s
        else:
            x=i-s
        y=j
        draw_line(x,y-s,x,y+d+s)
paint_buffer()
```

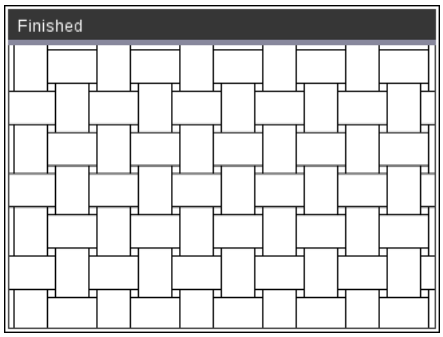


Voor de horizontale lijnstukken wisselen we de rollen van x en y om:

```
for j in range(0,212,d):
    for i in range(0,318,d):
        if (i+j)%2 == 0:
            y=j-s
        else:
            x=j+s
        x=i
        draw_line(x-s,y,x+ds,y)
paint_buffer()
```

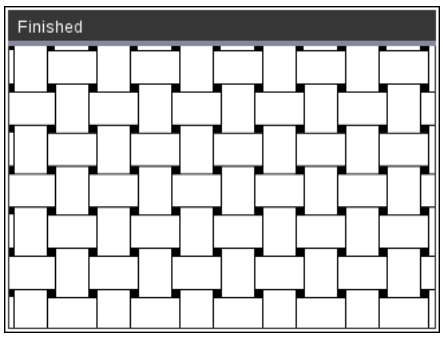


Beide blokken code voor de verticale en horizontale lijnstukken gecombineerd, geeft:



Rest nog het inkleuren van de vierkantjes met zijde 2*s:

```
for j in range(0,212,d):
    for i in range(0,318,d):
        fill_rect(i-s,j-s,2*s,2*s)
paint_buffer()
```



19.3. Verborgen cirkels

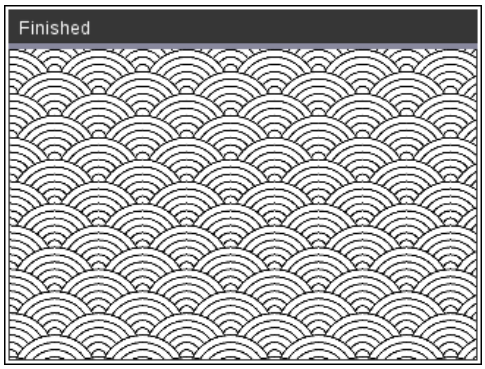
In het volgende voorbeeld wordt de kern van de herhalingen bepaald door zes cirkel met hetzelfde middelpunt en achtereenvolgend de stralen r = 30, r = 25, r = 20, r = 15, r = 10 en r = 5.

De x-coördinaten van de middelpunten laten we om de rij variëren tussen:

- even rij: x = 0, x = 60, x = 120, x = 180, x = 240, x = 330
- oneven rij: x = 30, x = 90, x = 150, x = 210, x = 270, x = 330

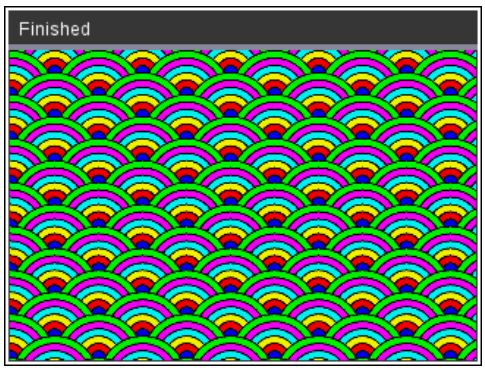
Voor de y-coordinaten verdelen we de hoogte van het scherm in 15.

```
from ti_draw import *
use_buffer()
# 15 rijen met figuren
for j in range(0,212/15+2):
    x=30*(j%2)
#   6 figuren per rij, niet altijd volledig zichtbaar
    for i in range(0,318,60):
#       Tekenen van 6 cirkels
        for r in range(30,0,-5):
            set_color(255,255,255)
            fill_circle(i+x,j*15,r)
            set_color(0,0,0)
            draw_circle(i+x,j*15,r)
paint_buffer()
```



Door set_color(255,255,255) te vervangen door set_color(75+6*r,75+6*r,75+6*r) worden er grayscale-schakeringen toegevoegd. Met het volgende kleurenpalet kan wat kleur aan de figuur toegevoegd worden.

```
from ti_draw import *
use_buffer()
def kleur(k):
    pal=[0,0,1,0,1,1,0,0]
    n=k%6
    set_color(240*pal[n],240*pal[n+1],240*pal[n+2])
k=0
for j in range(0,212/15+2):
    x=30*(j%2)
    for i in range(0,318,60):
        for r in range(30,0,-5):
            k=k+1
            kleur(k)
            fill_circle(i+x,j*15,r)
            set_color(0,0,0)
            draw_circle(i+x,j*15,r)
paint_buffer()
```



19.4. Optische misleidingen

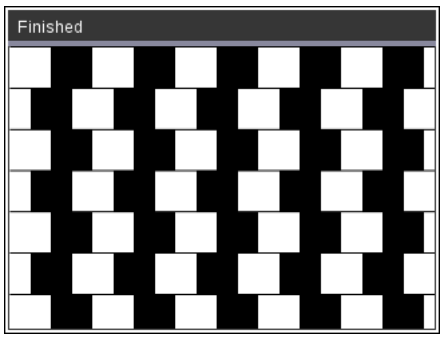
Voorbeeld 1
Parallelle vierkanten?

We coderen rijen van wit-zwarte vierkante (zijde 31):

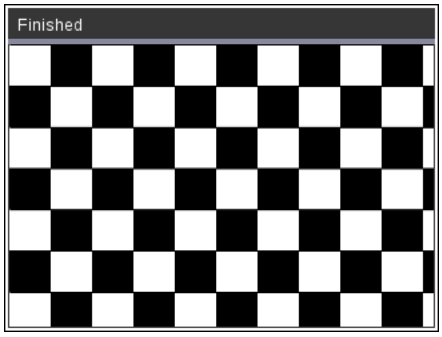
```
from ti_system import *.
from ti_draw import *
factor=float(input("Factor -1 ...  1: "))
use_buffer()
def vierkant(x,y,zijde):
    px=[x,x+z,x+z,x,x]
    py=[y,y,y+z,y+z,y]
    fill_poly(px,py)
z=31
set_color(255,255,255)
fill_rect(0,0,318,212)
set_color(0,0,0)
for j in range(8):
    draw_line(0,j*z,318,j*z)
    x=z
    if j%2 == 1:
        x=-factor*z
    for i in range(0,12,2):
        vierkant(i*z+x,j*z,z)
paint_buffer()
```

Merk op dat het input-statement in de code moet staan voor de eerste functie van de TI Draw-module, daar zo'n functie het grafische venster activeert.

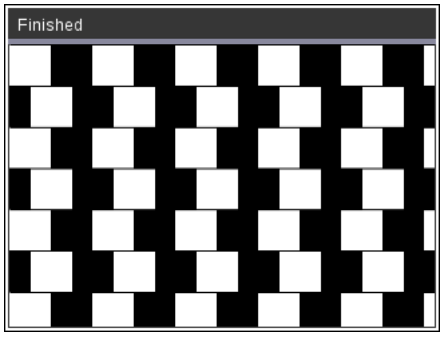
Factor = -0.5



Factor = 0



Factor = 0.5



Voorbeeld 2
Knipperende stippen?

De code voor het tekenen van het onderstaande raster bestaat uit de volgende delen:

Set up code en venster

```
from ti_draw import *
use_buffer()
zijde=40 ; b=3
fill_rect(0,0,318,212)
```

Horizontale lijnen (grijs)

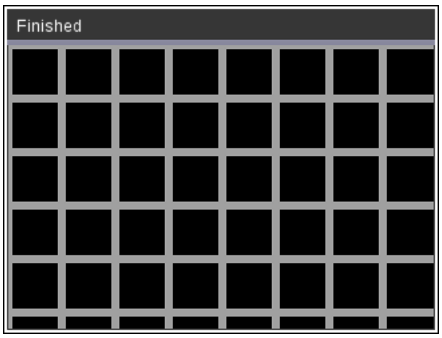
```
set_color(160,160,160)
for j in range(0,212,zijde):
    fill_rect(0,j-b,318,2*b)
```

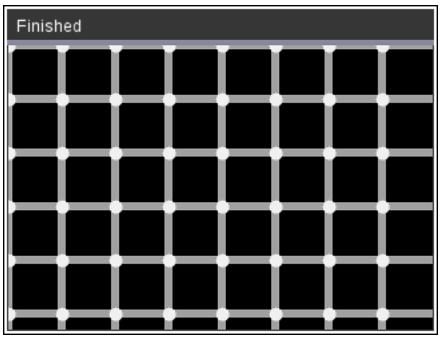
Verticale lijnen (grijs)

```
for i in range(0,318,zijde):
    fill_rect(i-b,0,2*b,212)
```

Stippen (wit)

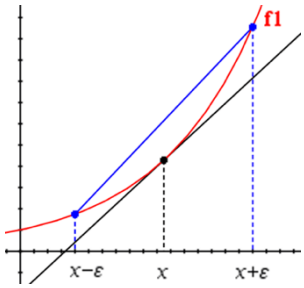
```
set_color(240,240,240)
for j in range(0,212,zijde):
    for i in range(0,318,zijde):
        fill_circle(i,j,1.7*b)
paint_buffer()
```





Programmeeropdrachten

Opdracht 1: Afgeleide functie

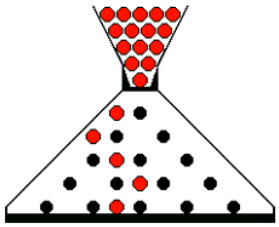


a. Plot de functie $f(x) = \sin(x)$ voor $x \in [-4\pi, 4\pi]$.

b. Plot in hetzelfde assenstelsel de afgeleide functie $f' = \frac{d}{dx}(f(x))$ voor $x \in [-4\pi, 4\pi]$, gebruikmakend van de numerieke afgeleide:

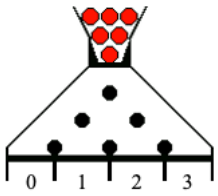
$$f'(x) = \frac{f(x+\varepsilon)-f(x-\varepsilon)}{2\varepsilon} \text{ met } \varepsilon = 0.003.$$

Opdracht 2: Kans-simulatie



Simuleer een Galton-bord met vijf rijen pinnen en plot de kansverdeling, samen met de numerieke gesimuleerde kansen.

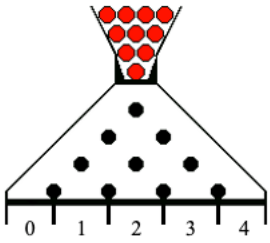
Wanneer een bal een pin raakt is de kans dat de bal links rolt gelijk aan de kans dat de bal rechts rolt.



Bijvoorbeeld voor een bord met drie rijen pinnen is/zijn er:

- 1 route tot uiterst links, slot 1: LLL,
- 3 routes tot slot 2: LLR, LRL en RLL,
- 3 routes tot slot 3: RRL, RLR en LRR
- 1 route tot slot 3: RRR

Het bord heeft een binomiale kansverdeling X ~ B(3,½): P(X=0) = $\frac{1}{8}$ P(X=1) = $\frac{3}{8}$ P(X=2) = $\frac{3}{8}$ P(x=3) = $\frac{1}{8}$



Een bord met vier rijen pinnen heeft een binomiale kansverdeling B(4,½):

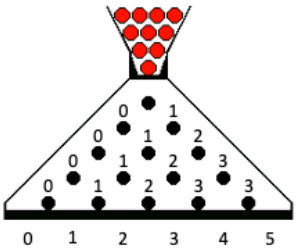
P(X=0) = $\frac{1}{16}$ P(X=1) = $\frac{4}{16}$ P(X=2) = $\frac{6}{16}$ P(x=3) = $\frac{4}{16}$ P(x=4) = $\frac{1}{16}$

Algemeen heeft een Galton-bord met n-rijen een binomiale kansverdeling B(n,½).

Voor de simulatie stellen we naar links vallen gelijk aan nul en naar rechts met 1.

Een simulatie van het rollen van één bal kan met de volgende code:

```
p=0
for i in range(5):
♦♦p=p+randint(0,1)
```



De waarde van p geeft aan in welk slot de bal terecht komt.

Opdracht 3: Lineaire regressie

Teken een puntenwolk en bepaal de regressierechte voor de onderstaande data, de geboortegewichten en lengtes van 15 baby's:

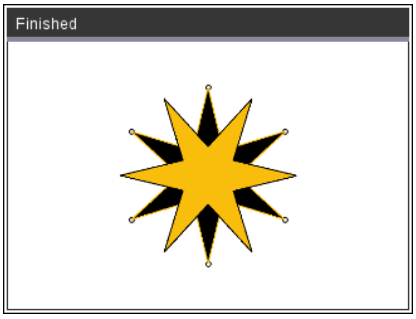
lengte = [53,52,53,56,51,51,54,53,50,53,50,52,53,52,52]
gewicht = [4.22,3.08,3.76,4.75,3.54,3.54,4.13,3.76,2.8,3.49,2.86,3.13,3.18,3.31,3.9]

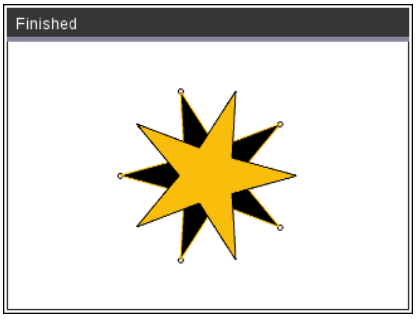
Opdracht 4: Fotobewerking

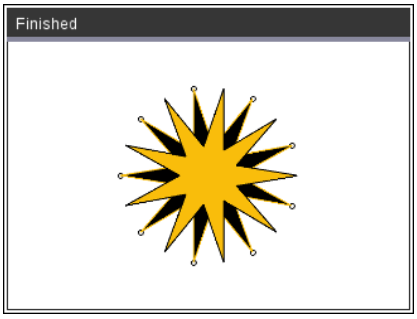
Upload je favoriete foto in een Notes-applicatie van de TI-Nsprire CX-software en pas een aantal manipulaties & filters uit de TI Python BootCamp toe.

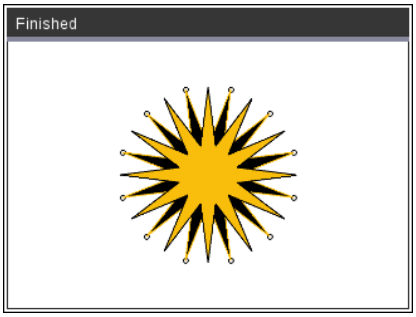
Opdracht 5: Ster

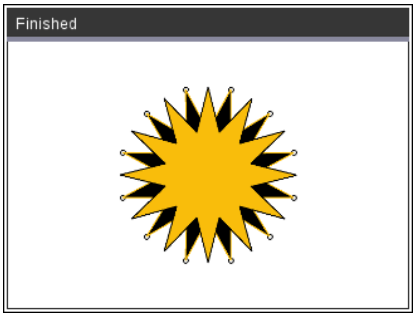
Teken/codeer de onderstaande ster en gebruik variabelen om de onderstaande varianten te genereren.

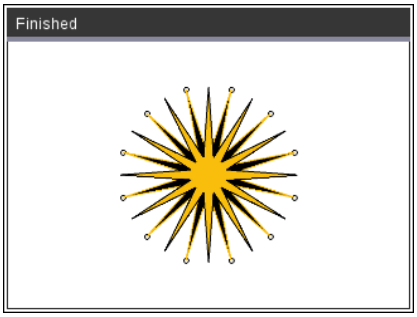


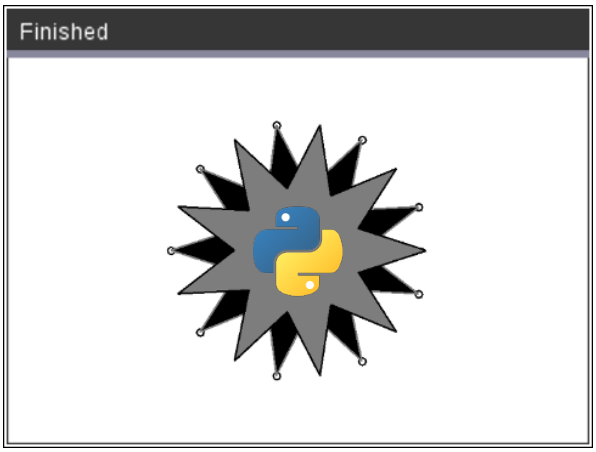












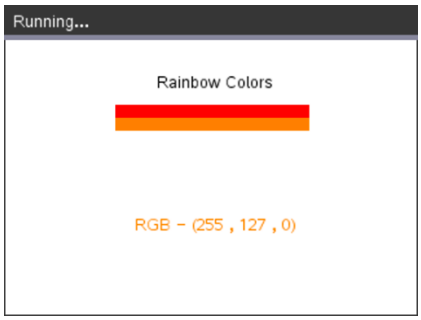
Verdieping

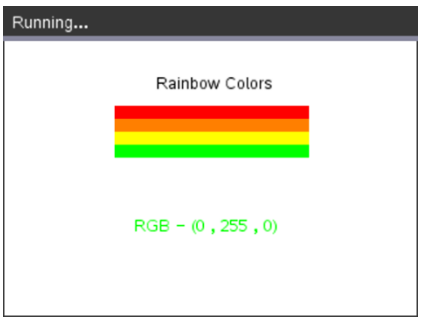
a. Dashboard

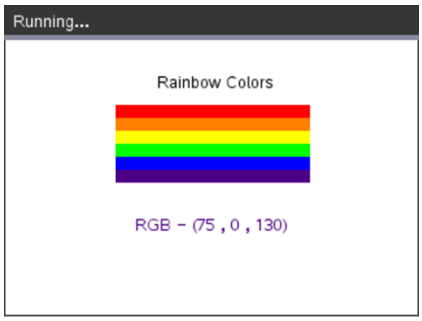
Gebruikmakend van de grafische modules kunnen we samen met het sturen en lezen van de TI-Innovator hub een grafisch dashboard coderen/genereren.

Een eerste voorbeeld is het aansturen van de ingebouwde RGB-led met de kleuren van de regenboog samen met het tonen van deze kleuren in het grafische shell-venster.

```
from ti_hub import *
from ti_draw import *

use_buffer()

# Rainbow colors
red=[255,255,255,0,0,75,143]
green=[0,127,255,255,0,0,0]
blue=[0,0,0,0,255,130,255]

set_color(0,0,0)
draw_text(117,40,"Rainbow Colors")

for i in range(7):
    color.rgb(red[i],green[i],blue[i])
    set_color(red[i],green[i],blue[i])
    fill_rect(85,50+10*i,150,10)
    draw_text(100,150,"RGB - ({} , {} , {})".format(red[i],green[i],blue[i]))
    paint_buffer()
    sleep(2)
    set_color(255,255,255)
    fill_rect(85,130,150,50)

color.off()
```







In een tweede voorbeeld gaat een SOS-alarmtoon klinken indien de helderheid, gemeten via de ingebouwde lichthelderheid-sensor, kleiner wordt dan een bepaalde waarde. Gelijktijdig verschijnt een visualisatie van de SOS-code in het grafische venster.

```
from ti_hub import *
from ti_image import *
from ti_draw import *

pic=load_image("SOS")
on=load_image("AlarmON")
off=load_image("AlarmOFF")

pic.show_image(25,10)
```

```
while get_key() != "esc":
    b=brightness.measurement()
    if b < 0.5:
        on.show_image(120,160)
        for i in range(3):
            set_color(0,0,0)
            fill_circle(40+i*25,150,5)
            sound.tone(100,0.2)
            sleep(0.2)
            sound.tone(0,0.2)
            sleep(0.2)
        for i in range(3):
            set_color(254,19,0)
            fill_rect(108+i*35,145,25,10)
            sound.tone(200,0.5)
            sleep(0.5)
            sound.tone(0,0.2)
            sleep(0.2)
        for i in range(3):
            set_color(0,0,0)
            fill_circle(220+i*25,150,5)
            sound.tone(100,0.2)
            sleep(0.2)
            sound.tone(0,0.2)
            sleep(0.2)
        sleep(0.3)
        set_color(255,255,255)
        fill_rect(30,145,250,18)
    else:
        off.show_image(120,160)
    sleep(0.1)
```

b. Keyboard-input & lezen van de muis-positie

i. Keyboard-input

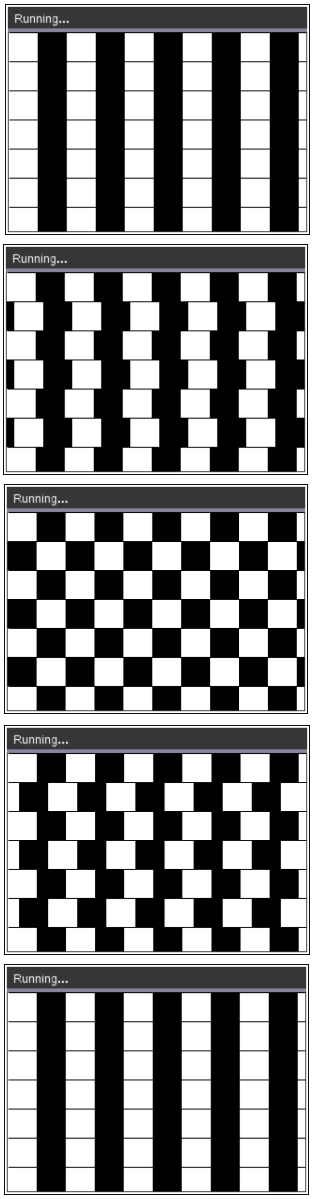
Voor het coderen van de onderstaande *Optische misleiding* veranderen we de variabele factor met behulp van het get_key()-statement (van de TI System-module) in een while-lus. We verminderen of vermeerderen factor met 0.05 door respectievelijk het intikken van ← en →.

De pijltjes-toetsen corresponderen in python met de volgende strings:

- ← “left”
- → “right”
- ↑ “up”
- ↓ “down”

Het teken van het patroon zelf, definiëren we als een functie.

```
from ti_system import *
from ti_draw import *
use_buffer()
def vierkant(x,y,zijde):
    px=[x,x+z,x+z,x,x]
    py=[y,y,y+z,y+z,y]
    fill_poly(px,py)
def tekenen():
    set_color(255,255,255)
    fill_rect(0,0,318,212)
    set_color(0,0,0)
    for j in range(8):
        draw_line(0,j*z,318,j*z)
        x=z
        if j%2==1:
            x=-factor*z
        for i in range(0,12,2):
            vierkant(i*z+x,j*z,z)
z=31 ; factor=0.5 ;  key=" "
while key!="esc":
    tekenen()
    paint_buffer()
    key=" "
    while key not in ("esc","left","right"):
        key=get_key()
    if key=="right":
        factor=max(-1,factor-0.05)
    if key=="left":
        factor=min(1,factor+0.05)
```



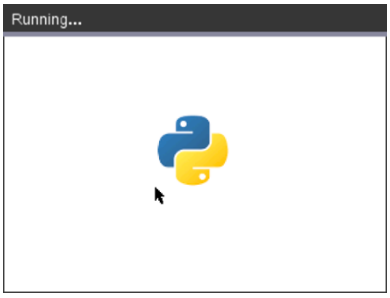
Merk op dat indien we het statement get_key(1) uitvoeren dat de code stopt met runnen tot dat een toets wordt ingedrukt.

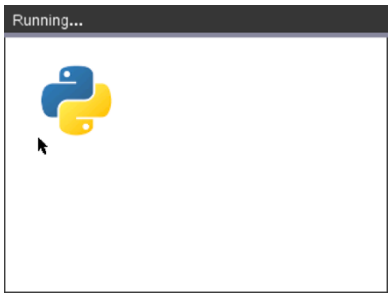
ii. Muis-positie

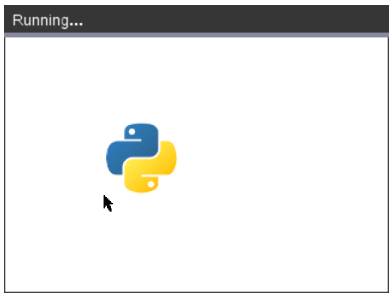
Met get_mouse() van de TI System-module lezen we de positie van de cursor. Het resultaat is een tuple met de x- en y-coördinaat van de positie. Indien de cursor buiten het venster valt, is de waarde van de coördinaten gelijk aan -1.

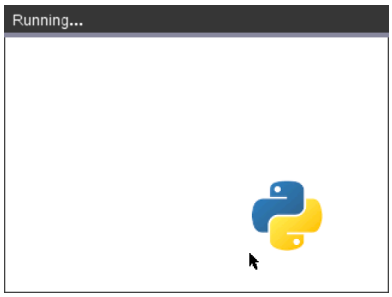
De onderstaande code toont het Python-logo met als coördinaten van de linkerbenedenhoek de positie van de muis; en dit totdat je klikt in het scherm. Merk op dat "center" overeenkomt met een muisklik en ook gecheckt kan worden met get_key().

```
from ti_image import *
from ti_system import *
img=load_image("py")
use_buffer()
while get_key()!="scherm":
    clear()
    x,y=get_mouse()
     img.show_image(x,y-img.h)
    paint_buffer()
```



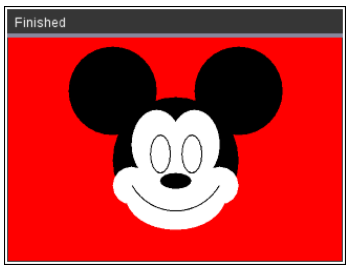






Voor de code van Mickey Mouse definiëren we het tekenen van Mickey als een functie, pupillen niet inbegrepen:

```
from ti_draw import *
from ti_system import *
dim=get_screen_dim()
set_window(-dim[0]/2,dim[0]/2,-dim[1]/2,dim[1]/2)
set_color(255,0,0)
fill_rect(-dim[0]/2,-dim[1]/2,318,212)
```

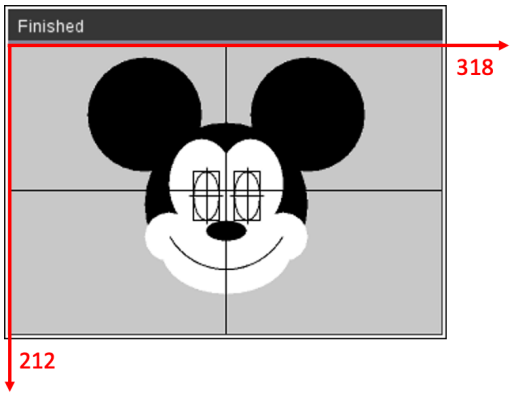



```
def mickey():
    set_color(255,0,0)
    fill_rect(-dim[0]/2,-dim[1]/2,318,212)
    set_color(0,0,0)
    fill_circle(0,-10,60)
    fill_circle(-60,56,42)
    fill_circle(60,56,42)
    set_color(255,255,255)
    fill_arc(-48,-76,96,66,0,360)
    fill_circle(42,-34,18)
    fill_circle(-42,-34,18)
    fill_arc(-42,-28,48,66,0,360)
    fill_arc(-6,-28,48,66,0,360)
    set_color(0,0,0)
    draw_arc(-24,-22,19,36,0,360)
    draw_arc(6,-22,19,36,0,360)
    fill_arc(-15,-37,30,15,0,360)
    draw_arc(-48,-58,96,96,210,120)
```

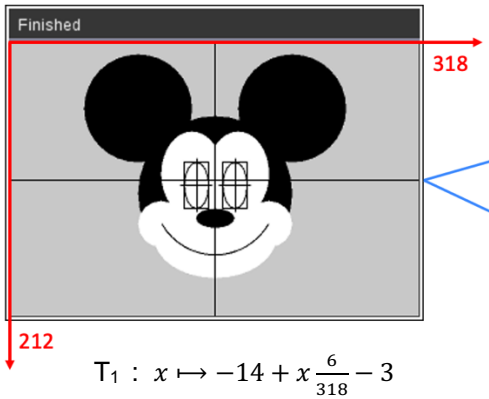
Met get_mouse() gaan we zorgen dat de pupillen de bewegingen van de cursor volgen.

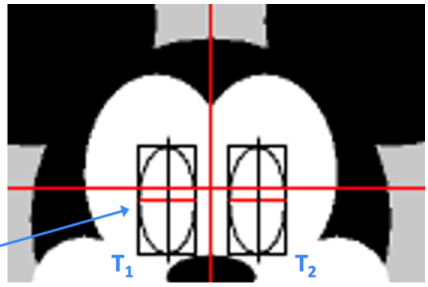
Een mogelijkheid is d.m.v. de onderstaande lineaire transformaties.

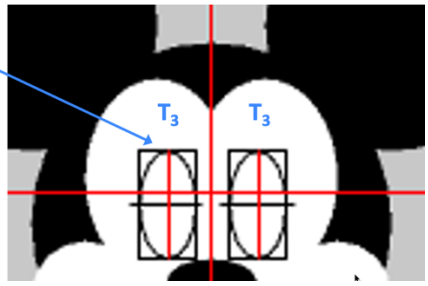
De coördinaten van de cursor bewegen zich in het venster (0,0) – (318,212) zoals hieronder aangegeven en Mickey leeft in het gedefinieerde venster: set_window(-dim[0]/2,dim[0]/2,-dim[1]/2,dim[1]/2).



We transformeren de coördinaten van de cursor met de lineaire transformaties T_1, T_2 en T_3 naar de loodlijnen door de middelpunten van de zijdes van de omschreven rechthoek van de bogen die de ogen van Micky voorstellen:







$$T_1 : x \mapsto -14 + x\frac{6}{318} - 3$$

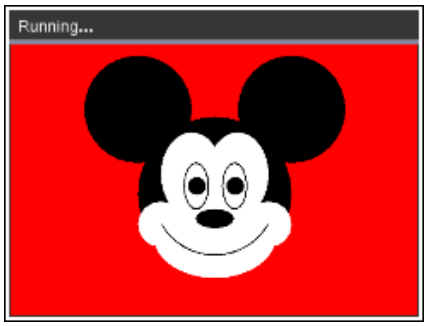
$$T_2 : x \mapsto 16 + x\frac{6}{318} - 3$$

$$T_3 : y \mapsto -4 + (212 - y)\frac{22}{212} - 11$$

De lineare transformaties T_1 en T_2 beelden we het segment [0,318] af op respectievelijk [-17,-11] en [13,19].

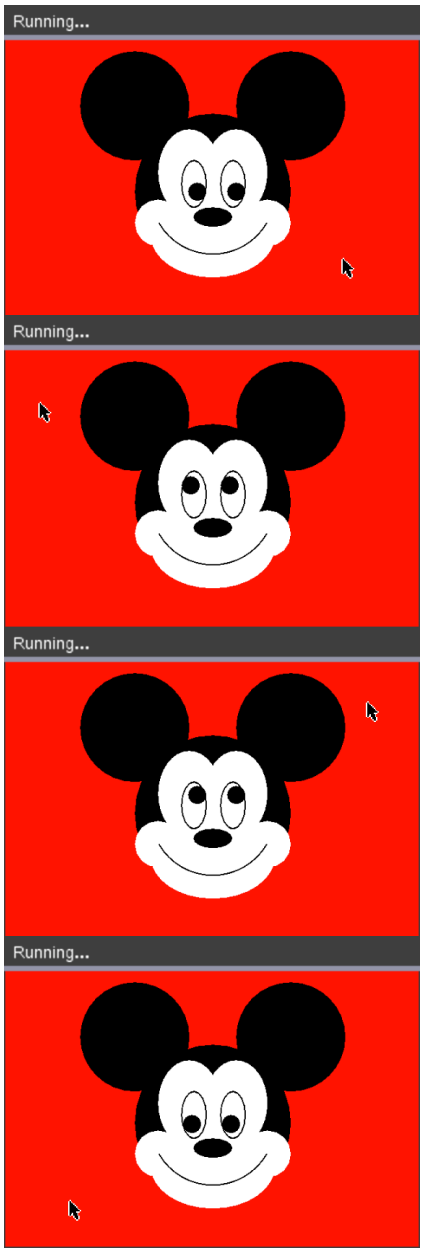
En T_3 beeldt het segment [0,212] af op [-15,7], samen met het veranderen van de positieve oriëntatie.

Voor iedere iedere positie van de cursor bekomen we zo een pixel in ieder oog van Mickey waar we telkens een gevulde cirkel tekenen met straal van een zevental pixels.



We vervolgen het programma als volgt:

```
use_buffer()
while get_key() != "center":
    clear()
    mickey()
    xm,ym=get_mouse()
    if xm == -1:
        n=0
    else:
        n=xm*6/318-3
    if ym == -1:
        m=0
    else:
        m=(212-ym)*22/212-11
    fill_circle(-14+n,-4+m,7)
    fill_circle(16+n,-4+m,7)
    paint_buffer()

    mickey()
    fill_circle(15,7.3,7)
    fill_circle(-14,7.3, 7)
    paint_buffer()
```



iii. Een digitale analoge klok

Gebruikmakend van o.a. de time-module coderen we een analoge klok.

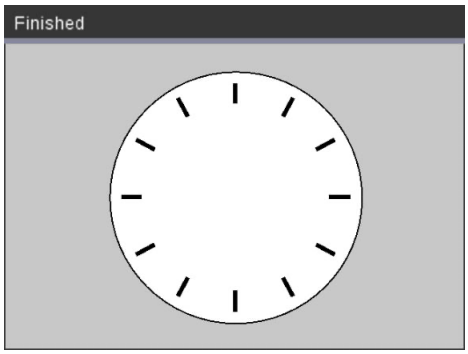
Het statement localtime() van de time-module creëert een tuple met de volgende data:
(jaar , maand , dag , uur, minuut , seconde , weekdag , dagnummer, Daylight Savings indicator).

```
Python Shell
>>>from time import *
>>>localtime()
(2020, 11, 26, 22, 7, 50, 3, 331, 0)
>>>
```

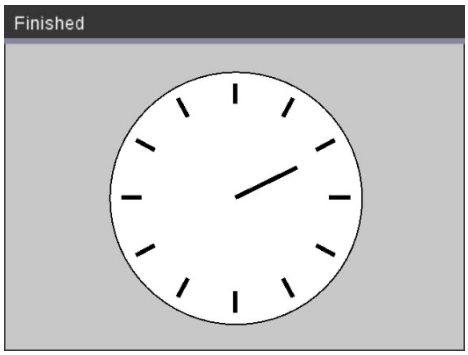
Samen met de onderstaande grafische definities bouwen we digitaal een analoge klok.

```
from math import *
from time import *
from ti_draw import *
from ti_system import *
set_window(-3,3,-2,2)
r=1.5
use_buffer()
def draw_clock():
    set_color(200,200,200)
    fill_rect(-3,-2,6,4)
    set_color(255,255,255)
    fill_circle(0,0,1.1*r)
    set_color(0,0,0)
    draw_circle(0,0,1.1*r)
    set_pen(1,0)
    set_color(0,0,0)
    for i in range(12):
        a=pi/2-i*pi/6
        x1=0.82*r*cos(a)
        y1=0.82*r*sin(a)
        x2=1.0*r*cos(a)
        y2=1.0*r*sin(a)
        draw_line(x1,y1,x2,y2)

def hour_hand(h,m):
    a=pi/2-h*pi/6-m*pi/360
    x=0.6*r*cos(a)
    y=0.6*r*sin(a)
    draw_line(0,0,x,y)
```

draw_clock()

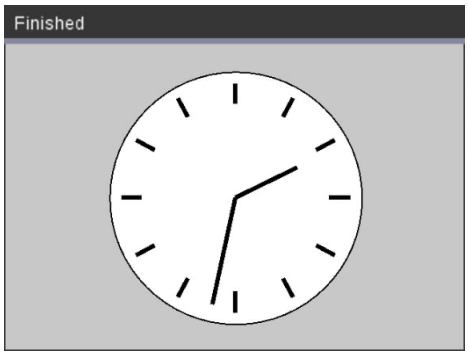


hour_hand(14,11)



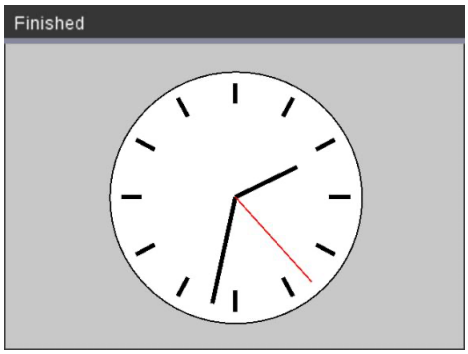
```
def minute_hand(m):
    a=pi/2-m*pi/30
    x=0.95*r*cos(a)
    y=0.95*r*sin(a)
    draw_line(0,0,x,y)
```

minute_hand(32)

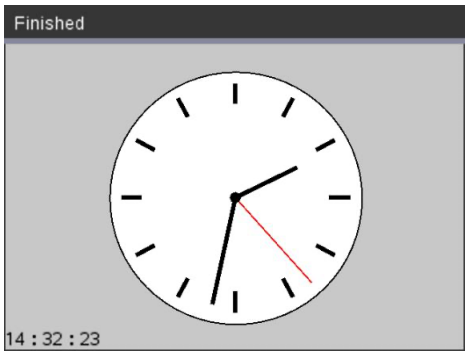



```
def second_hand(s):
    set_color(255,0,0)
    a=pi/2-s*pi/30
    x=r*cos(a)
    y=r*sin(a)
    draw_line(0,0,x,y)
```

second_hand(23)




```
while get_key() != "esc":
    clear()
    draw_clock()
    t=localtime()
    hr,min,sec=t[3],t[4],t[5]
    draw_text(-3,-2,"{} : {} : {}".format(hr,min,sec))
    set_color(0,0,255)
    hour_hand(hr,min)
    minute_hand(min)
    set_color(255,0,0)
    set_pen(0,0)
    second_hand(sec)
    set_color(0,0,0)
    fill_circle(0,0,0.05*r)
    paint_buffer()
```



Een uniforme lay-out van de tijd in de linkerbenedenhoek met 6 digits kan met de volgende functie:

```
def tijd():
    t = localtime()
    hr = ("0"+str(t[3]))[-2:]
    min = ("0"+str(t[4]))[-2:]
    sec = ("0"+str(t[5]))[-2:]
    return hr + " : " + min + " : " + sec
```

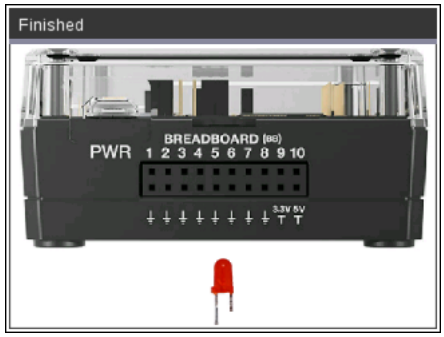
20. Virtuele LEDs

Met de grafische modules en het definiëren van klassen, kunnen we op een vrij makkelijke en gebruiksvriendelijke manier STEM-experimenten simuleren. We bekijken twee voorbeelden: LEDs en een RGB-array.

20.1. Knipperend LED

We starten met het bepalen van de achtergrond. Voor dit voorbeeld een afbeelding van de BB-voorkant van de TI-Innovator™ hub en een afbeelding van een LED. We gebruiken hiervoor de functie background().

```
from ti_draw import *
from ti_image import *
def background():
    hub = load_image("hub")
    hub.show_image(0,0)
    led = load_image("led")
    led.show_image(150,160)
```



OFF

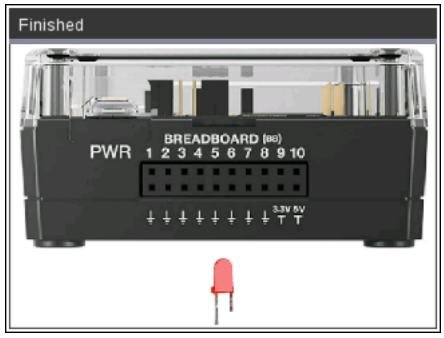
De LED definiëren we als een klasse met als attributen:

- de bijhorende figuur
- de x- en y-coördinaat waar de figuur te tonen

```
class Led():
    def __init__(self):
        self.img = load_image("led")
        self.x = 150
        self.y = 160
```

Als methodes coderen we de functies on() en off(); functies die ook beschikbaar in de TI Hub-modules.

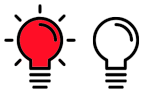
```
    def on(self):
        set_color(255,0,150)
        fill_rect(self.x+3,self.y+5,13,20)
        fill_circle(self.x+9,self.y+6,6)

    def off(self):
        self.img.show_image(self.x,self.y)
```

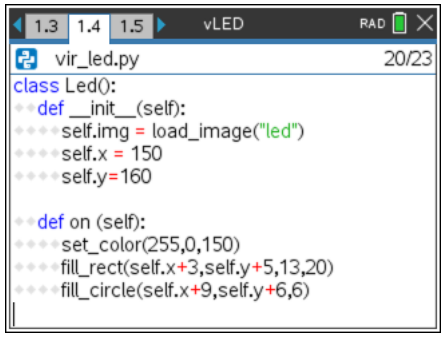


ON

Bovenstaande code bewaren we als het programma vir_led.py zodat we de objecten van de klasse Led() kunnen gebruiken in andere programma's, b.v. voor het knipperen van een LED:

```
from vir_led import *
from ti_system import *
from time import *

use_buffer()

background()
paint_buffer()

led = Led()

while get_key() != "esc":
    led.on() ; paint_buffer()
    sleep(0.3)
    led.off() ; paint_buffer()
    sleep(0.3)
```

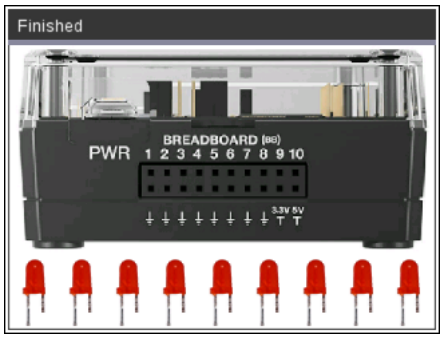




20.2. Lopend LED

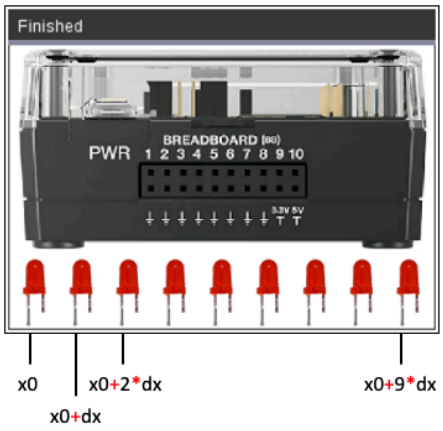
In het volgende voorbeeld gebruiken we 9 LEDs en breiden de achtergrond als volgt uit:

```
def background():
    hub = load_image("hub")
    led = load_image("led")
    hub.show_image(0,0)
    for i in range(9):
        led.show_image(10+i*35,160,160)
```



We passen de attributen van de klasse Led() aan zodat iedere LED bepaalt wordt door een BB-poort: “BB 1” tot en met “BB 9”.

```
class Led():
# port = “BB 1”, “BB 2”, ... of “BB 9”
    def __init__(self,port):
        self.port=port
        self.factor = int(self.port[3])
        self.img = load_image("led")
        self.dx = 35
        self.x0 = 10
        self.x = self.x0+(self.factor-1)*self.dx
        self.y = 160
```

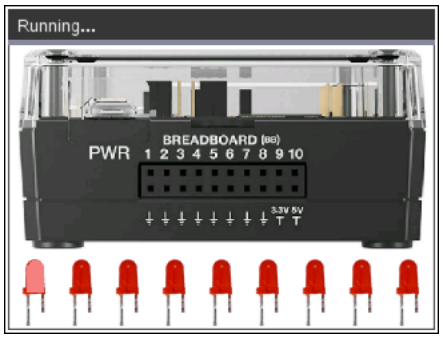


De methodes blijven hetzelfde en we bewaren deze codes als vir_hub.py.

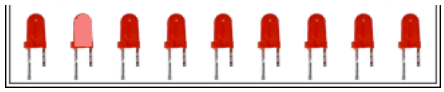
```
    def on(self):
        set_color(255,0,150)
        fill_rect(self.x+3,self.y+5,13,20)
        fill_circle(self.x+9,self.y+6,6)

    def off(self):
        self.img.show_image(self.x,self.y)
```

Een lopend LED coderen we met de klasse Led() als volgt:

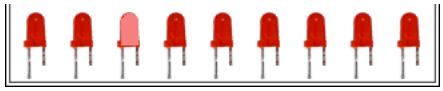
```
from vir_hub import *
from ti_system import *
from time import *

use_buffer()

background()
paint_buffer()

leds = []
for n in range(9):
    leds.append (Led("BB " + str(n+1)))

while get_key() != "esc":
    for vled in leds:
        vled.on() ; paint_buffer() ; sleep(0.2)
        vled.off() ; paint_buffer() ; sleep(0.2)
```



led = LED(“BB 1”) ; led.on()



led = LED(“BB 2”) ; led.on()

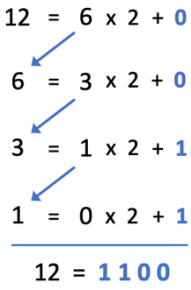


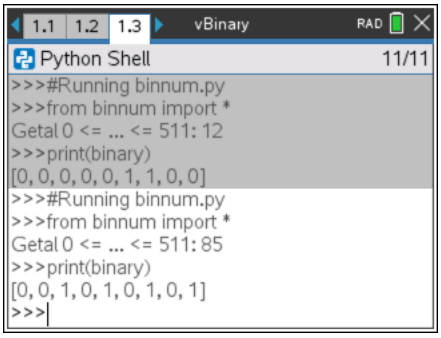
led = LED(“BB 3”) ; led.on()

20.3. Binaire LED-voorstelling

We definiëren eerst de functie dec2bin() voor de conversie van een natuurlijk getal, tussen 0 en 511, naar het corresponderende binair getal; telkens bestaande uit 9 digits.

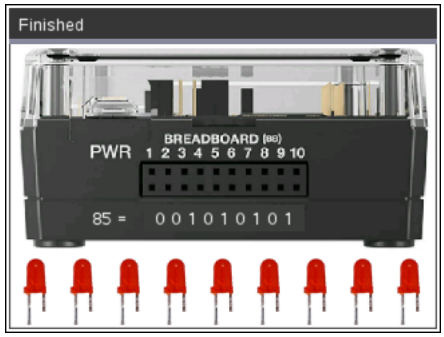
```
from vir_hub import *
from ti_system import *
from time import *

def dec2bin(t):
    bin=[]
    while t//2 != 0:
        bin.append(t%2)
        t=t//2
    bin.append(t%2)
# Aanvullen met 0'en tot 9 digits
    for i in range(0,9-len(bin)):
        bin.append(0)
    bin.reverse()
    return bin

n=int(input("Getal 0 <= ... < = 511: "))
binary=dec2bin(n)
```

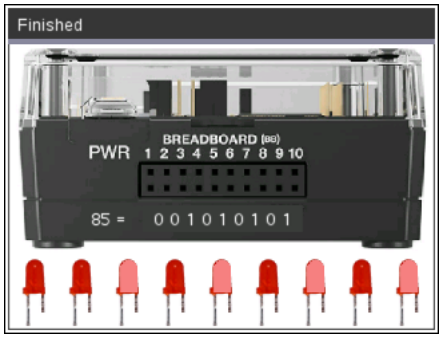




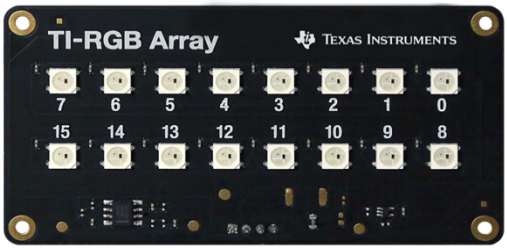
Met onderstaande code en gebruikmakend van vir_hub.py, printen we de binaire voorstelling en visualiseren we deze voorstelling m.b.v. de 9 LEDs.

```
background()

# Printen van de binaire voorstelling van n
set_color(57,60,57)
fill_rect(100,120,120,20)
set_color(175,175,175)
draw_text(62,138,"{} = ".format(n))
for i in range(len(binary)):
    draw_text(109+i*12,138,binary[i])
paint_buffer()

sleep(2)

# Binaire visualisatie met de negen Leds
for i in range(9):
    if binary[i] == 1:
        binled=Led("BB "+str(i+1))
        binled.on()
paint_buffer()
```





20.4. Virtuele RGB-array



Voor de TI-Innovator™ Hub is een TI-RGB array beschikbaar met 16 programmeerbare LEDs.

In de module TI Hub is een klasse rgb_array() beschikbaar. Voor een object uit deze klasse zijn o.a. de volgende methodes beschikbaar:

- set(led_positie,rood,groen,blauw)
- set_all(rood,groen,blauw)
- all_off()

We simuleren deze TI-RGB array als volgt.

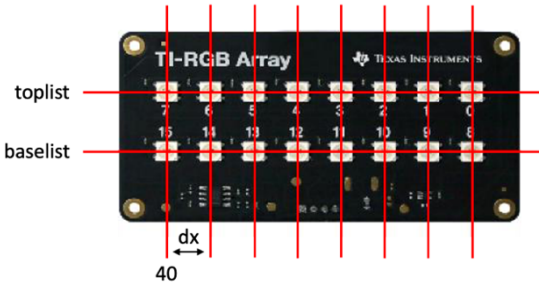
Als achtergrond gebruiken we een afbeelding van de TI RGB array.

```
from ti_image import *
from ti_draw import *
from time import *

def background():
    rgb=load_image("rgb")
    rgb.show_image(0,0)
```

Voor de attributen voor de klasse Array() definiëren we hoofdzakelijk lijsten met de coördinaten van de middelpunten van de cirkels die de LEDs simuleren:

- toplist — middelpunten LEDs: 0-7
- baselist — middelpunten LEDs: 8-15
- leds — middelpunten LEDs: 0-15
- topoff — simulatie off van LEDs 0-7
- baseoff — simulatie off van LEDs 8-15
- dx — afstand tussen LEDs
- rad — straal LEDs
- image — bijhorende figuur

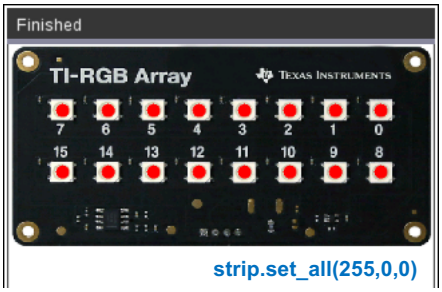
```
class Array:
    def __init__(self):
        self.dx=34
        self.toplist=[[40+i*self.dx,49] for i in range(8)]
        self.baselist=[[40+i*self.dx,95] for i in range(8)]
        self.leds=self.toplist+self.baselist
        self.topoff=[[41+i*self.dx,48] for i in range(8)]
        self.baseoff=[[42+i*self.dx,95] for i in range(8)]
        self.image=load_image("rgb")
        self.rad=5
```



Als methodes coderen we de volgende functionaliteit; ook beschikbaar in de module TI Hub.

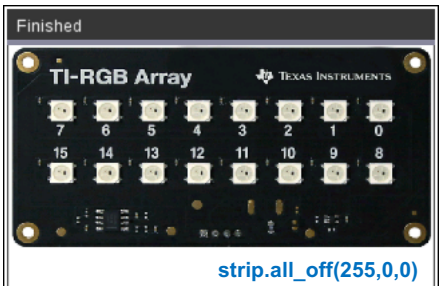
```
    def set(self,p,r,g,b):
        set_color(r,g,b)
        use_buffer()
        if p<=7:
            fill_circle(self.toplist[7-p][0],self.toplist[7-p][1],self.rad)
        else:
            fill_circle(self.baselist[15-p][0],self.baselist[15-p][1],self.rad)
        paint_buffer().
```

```
    def set_all(self,r,g,b):
        set_color(r,g,b)
        use_buffer()
        for i in range(16):
            fill_circle(self.leds[i][0],self.leds[i][1],self.rad)
        paint_buffer()
```



strip.set_all(255,0,0)

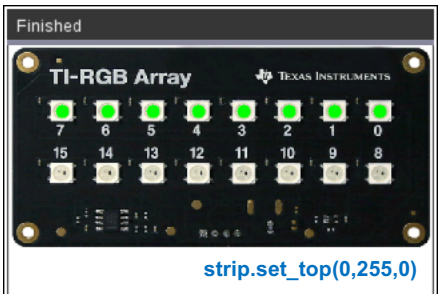
```
    def all_off(self):
        self.image.show_image(0,0)
```



strip.all_off(255,0,0)

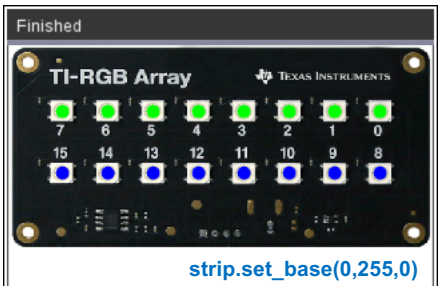
Als extra functionaliteit voegen we de volgende methodes toe en bewaren alle code in vir_rgb.py.

```
    def set_top(self,r,g,b):
        set_color(r,g,b)
        use_buffer()
        for i in range(8):
            fill_circle(self.toplist[i][0],self.toplist[i][1],self.rad)
        paint_buffer()
```



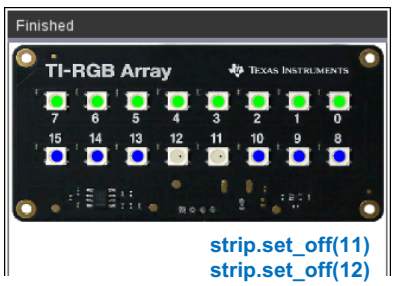
strip.set_top(0,255,0)

```
    def set_base(self,r,g,b):
        set_color(r,g,b)
        use_buffer()
        for i in range(8):
            fill_circle(self.baselist[i][0],self.baselist[i][1],self.rad)
         paint_buffer()
```



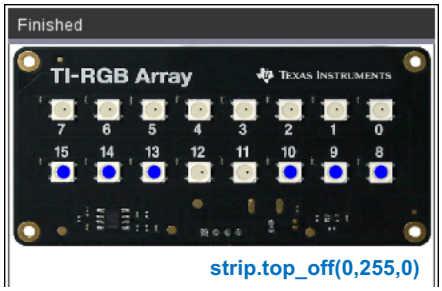
strip.set_base(0,255,0)

```
    def set_off(self,p):
        set_color(217,214,187)
        use_buffer()
        if p<=7:
            fill_circle(self.toplist[7-p][0],self.toplist[7-p][1],self.rad)
            set_color(0,0,0)
            fill_circle(self.topoff[7-p][0],self.topoff[7-p][1],1)
        else:
            fill_circle(self.baselist[15-p][0],self.baselist[15-p][1],self.rad)
            set_color(0,0,0)
            fill_circle(self.baseoff[15-p][0],self.baseoff[15-p][1],1)
        paint_buffer()
```



strip.set_off(11)
strip.set_off(12)

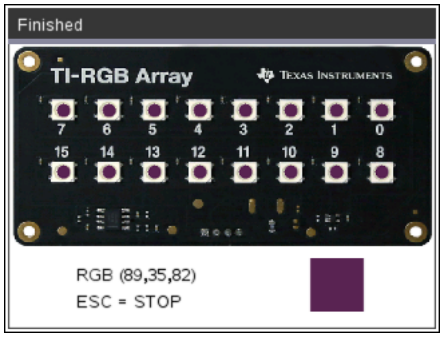
```
    def top_off(self):
        set_color(217,214,187)
        use_buffer()
        for i in range(8):
            fill_circle(self.toplist[i][0],self.toplist[i][1],self.rad)
        set_color(0,0,0)
        for i in range(8):
            fill_circle(self.topoff[i][0],self.topoff[i][1],1)
        paint_buffer()
```

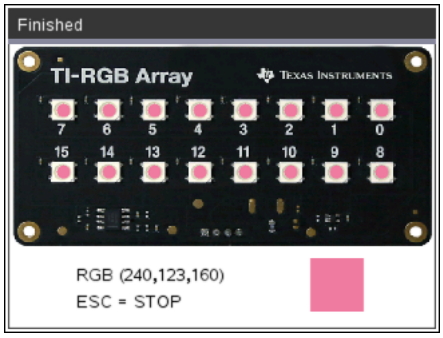



```
    def base_off(self):
        set_color(217,214,187)
        use_buffer()
        for i in range(8):
            fill_circle(self.baselist[i][0],self.baselist[i][1],self.rad)
        set_color(0,0,0)
        for i in range(8):
            fill_circle(self.baseoff[i][0],self.baseoff[i][1],1)
        paint_buffer()
```

We eindigen met de onderstaande at random kleurenanimatie.

```
from vir_rgb import *
from ti_system import *
from random import *

background()
strip=Array()

use_buffer()

while get_key() != "esc":
    r=randint(0,255)
    g=randint(0,255)
    b=randint(0,255)
    set_color(r,g,b)
    strip.set_all(r,g,b)
    fill_rect(225,160,40,40)
    set_color(255,255,255)
    fill_rect(50,160,150,50)
    set_color(0,0,0)
    draw_text(50,180,"RGB ({},{},{})".format(r,g,b))
    draw_text(50,200,"ESC = STOP")
    paint_buffer()
    sleep(2)
```





20.5. To Hub, or not to Hub0

Met de try: en except: statements kunnen we de code laten detecteren of de TI-Innovator™ Hub is aangesloten of niet.

Het try-blok zal een exception genereren indien de hub niet is aangesloten en dan wordt het except-blok uitgevoerd.

Indien de hub is aangesloten coderen we dat de module TI Hub wordt ingeladen en indien niet zelf gedefinieerde functionaliteit die een hub-experiment simuleert.

Als voorbeeld bekijken we een knipperend led.

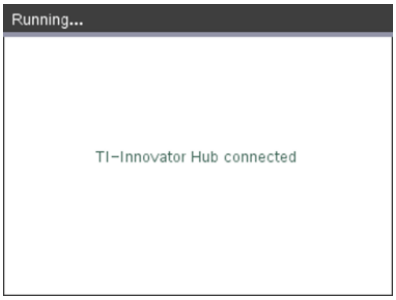
Stap 1	het try-blok controleert de connectie met een TI-Innovator Hub
Stap 2	if connected >> ti_hub import else >> vir_led import
Stap 3	uitvoeren experiment

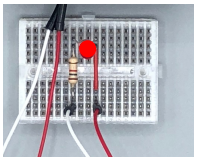
Dit geeft het volgende voorbeeld.

```
from ti_system import *
from ti_draw import *
from time import *

set_color(255,0,0)
draw_text(45,106,"Checking connection TI-Innovator Hub")

try:
    from ti_hub import *
```

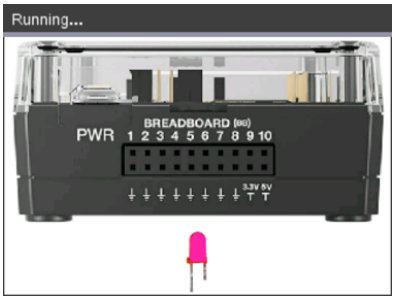
```
# TI-Innovator Hub connected
    clear()
    set_color(52,91,77)
    draw_text(75,106,"TI-Innovator Hub
connected")
    myled=led("BB 1")
```

```
# No TI-Innovator Hub connected
except:
    from vir_led import *
    background()
    myled=Led("BB 1")
```



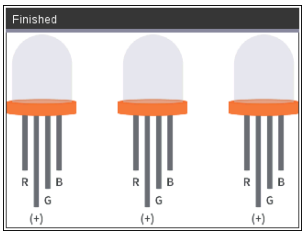


```
while get_key() != "esc":
    myled.on()
    sleep(0.3)
    myled.off()
    sleep(0.3)
```



Programmeeropdrachten

Opdracht 1: Een virtuele RGB-LED



"OUT1" "OUT 2" "OUT3"

Ontwikkel het volgende virtuele RGB-led-experiment.

a. Download een figuur van een RGB-led en creëer een background met driemaal de figuur van de RGB-led.

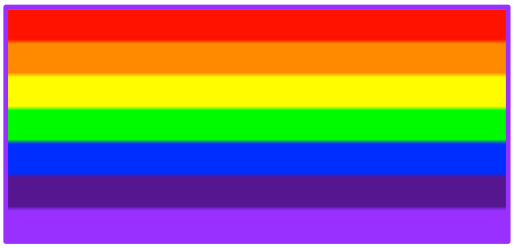
b. Definieer een klasse RGB() met als argument de poort , "OUT 1", "OUT 2", "OUT 3", die aangeeft welke figuur er bij een object van de klasse RGB correspondeert, respectievelijk links, midden of rechts.

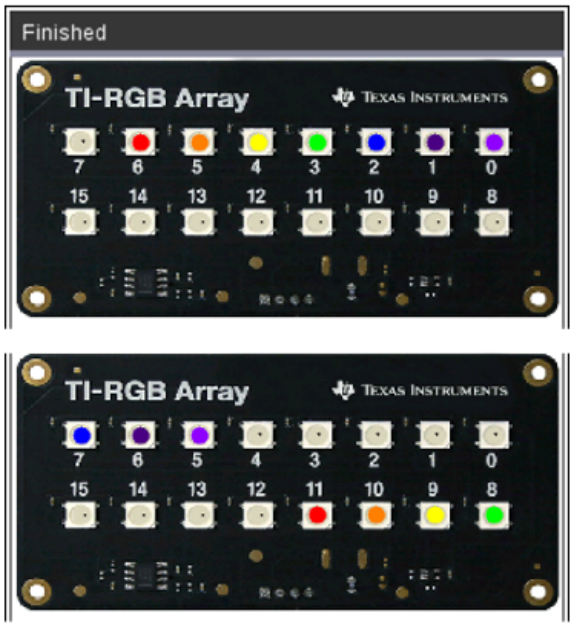
c. Codeer de methode set() die iedere led apart een rgb-kleur geeft en de methode off() die iedere led kan uitzetten.

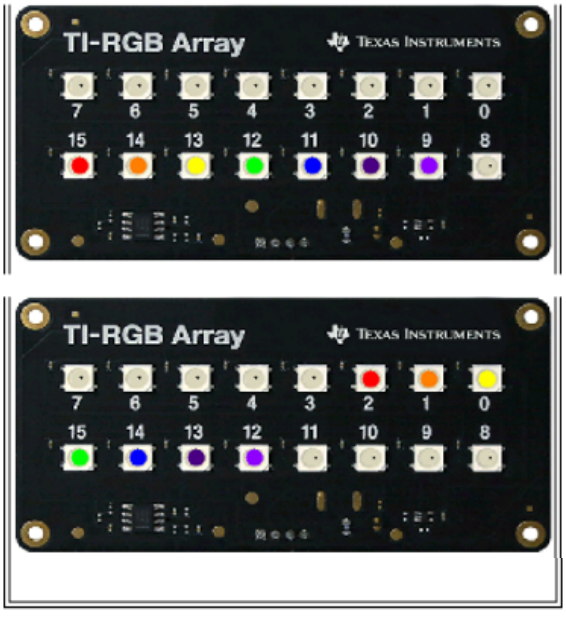
Opdracht 2: Een kleurig looplicht voor de virtuele RGB-array

Genereer met de module vir_rgb.py (zie BootCamp Deel 5) een gekleurd looplicht, b.v. met de kleuren van de regenboog. Voor de kleuren van de regenboog kan gebruik gemaakt worden van een lijst met de rgb-kleuren:

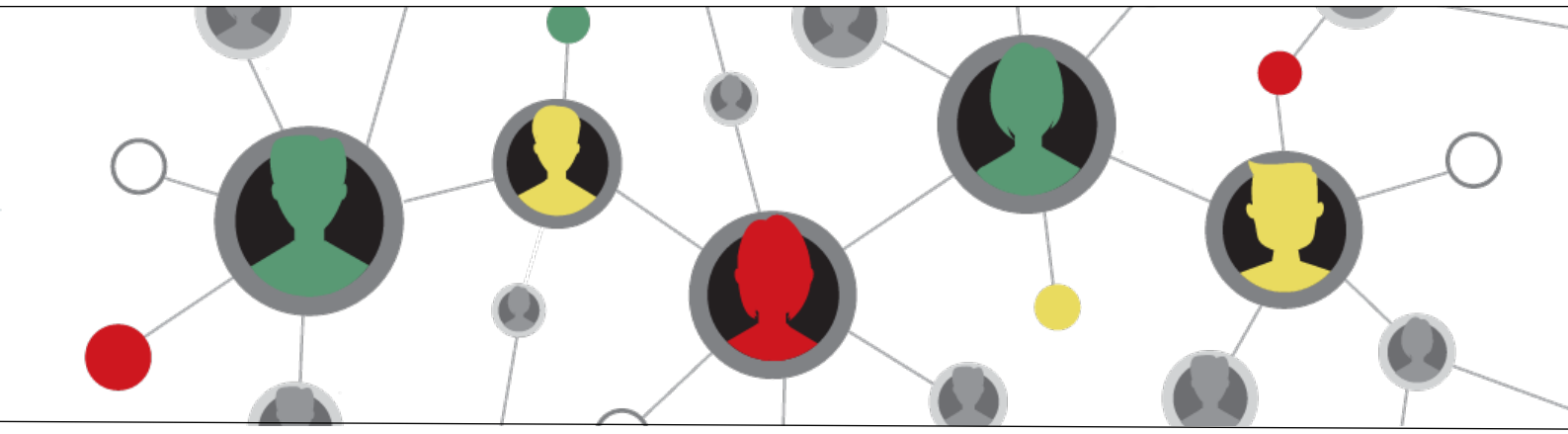
[[255,0,0],[255,127,0],[255,255,0],[0,255,0],[0,0,255],[75,0,130],[143,0,255]]







Als uitbreiding, voeg een parameter toe die de lengte van het looplicht (het aantal LEDs) aanpast.



T³ NEDERLAND T³ VLAANDEREN

www.wil-depython.be
www.wil-depython.nl

Texas Instruments