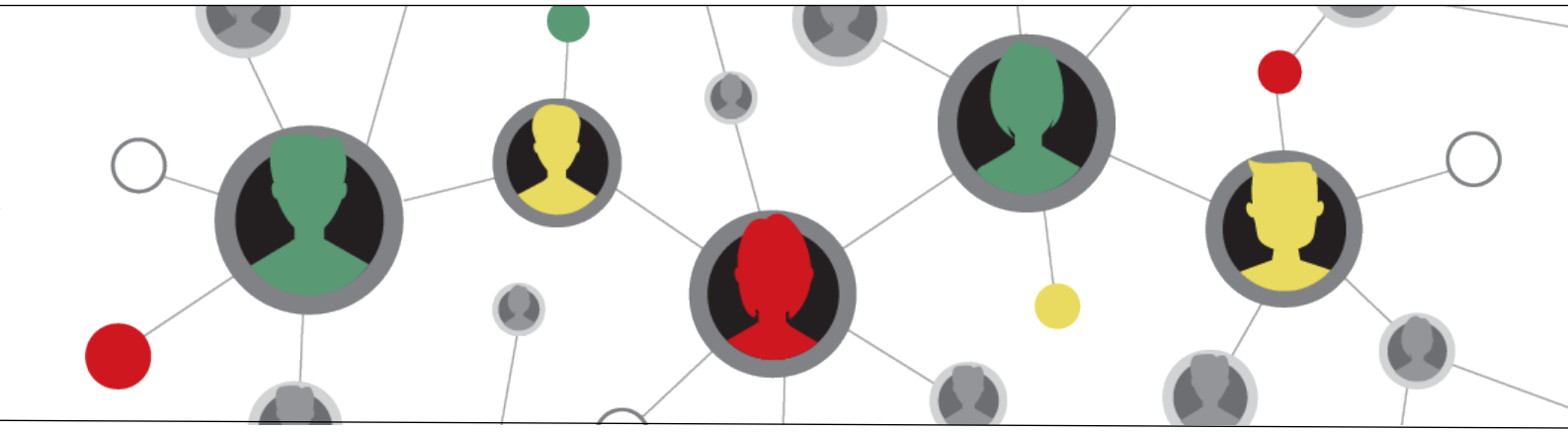
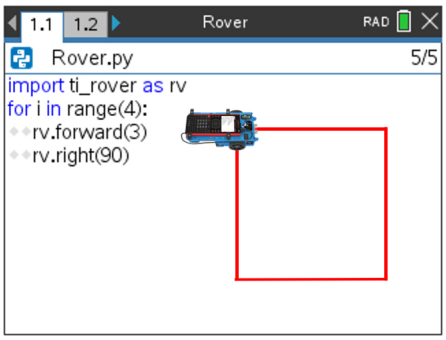
TI Python BootCamp

Deel 3 – Programmeren van een MCU



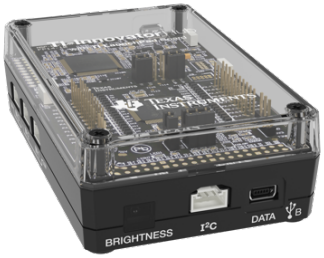


T³ NEDERLAND

Teachers Teaching with Technology™

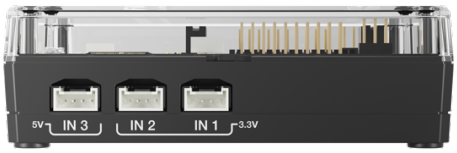
T³ VLAANDEREN

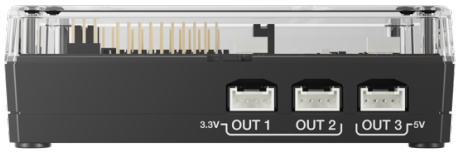
1. TI-Innovator™ Hub



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.

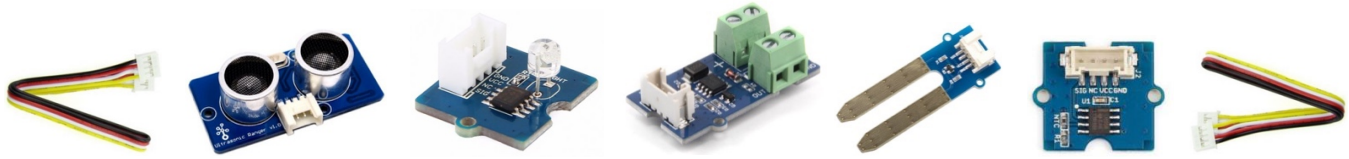
Door een BoosterPack wordt de TI-Innovator Hub uitgerust met 3 input-poorten en 3 output-poorten voor Grove-apparaten, waaronder tal van sensoren.





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

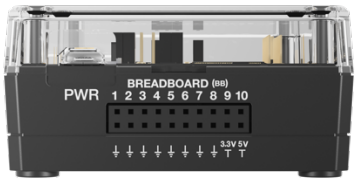
Afstand **Licht** **MosFet** **Vochtigheid** **Temperatuur**

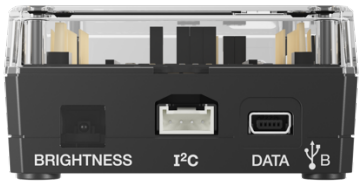


Enkele electronics online stores waar Grove-producten beschikbaar zijn in de Benelux:

- 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).





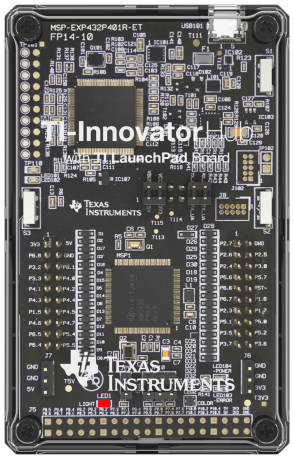
De DATA-poort (mini USB B) wordt gebruikt om de hub te verbinden met TI-handhelds en computers (Windows & Mac). Via de Micro-USB PWR-poort kan extra vermogen toegevoegd worden met b.v. een powerbank.

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

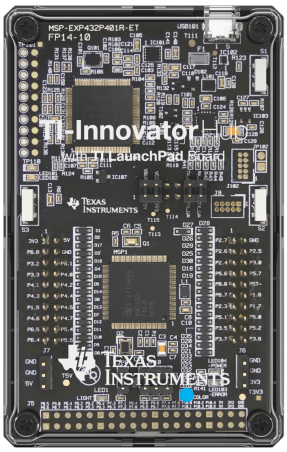
De TI-Innovator Hub is een microcontroller-systeem dat klaar is om op een eenvoudige manier te gebruiken in de klas en om nieuwgierigheid om te zetten in het begrijpen van wetenschappelijke concepten.

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 de programmeertalen TI Python en TI-Basic.

LED

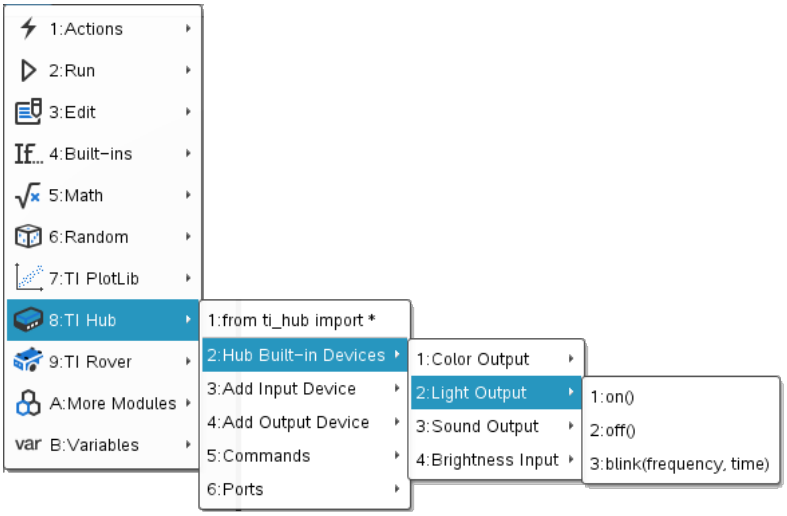


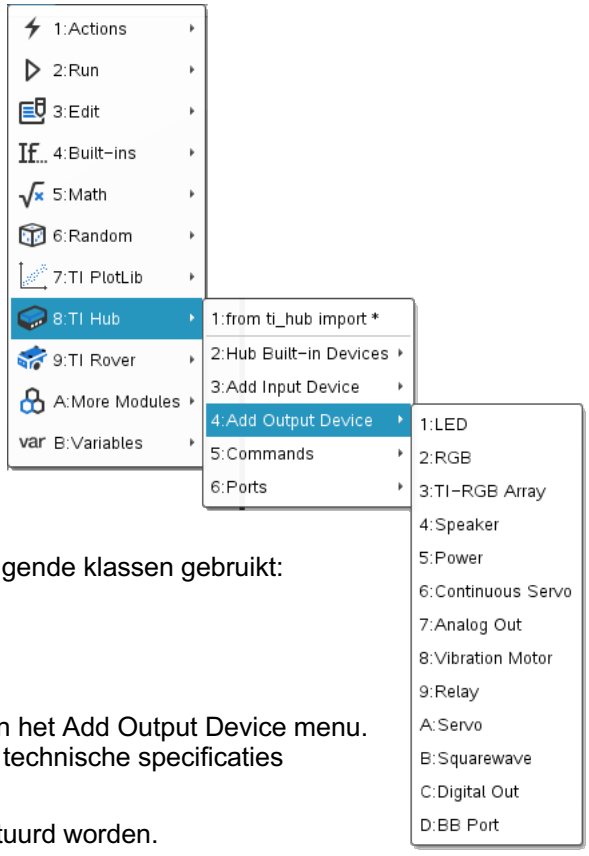
RGB LED



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

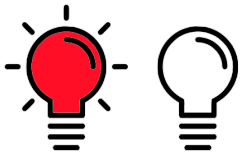
```
from ti_hub import *
light.on()
light off()
```

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()
```

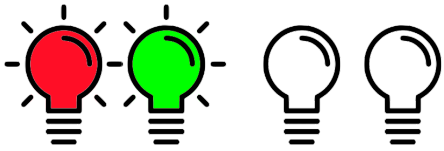
Indien we de led 5-keer aan en uit willen gaan, zetten we de bovenstaande code in een for-lus:

```
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:

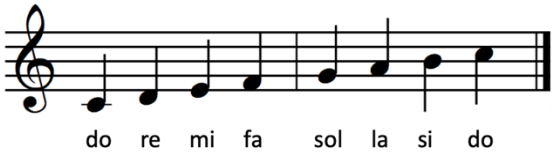
```
from ti_hub import *
sound.tone(250,2)
sleep(2)
sound.tone(500,2)
```

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)
```

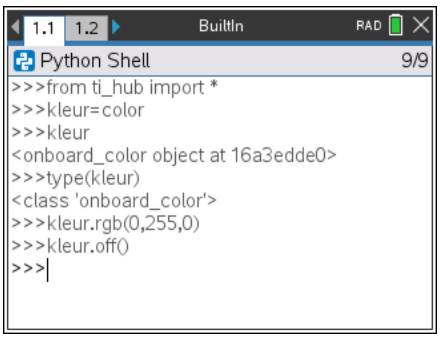
Of een toonladder spelen:

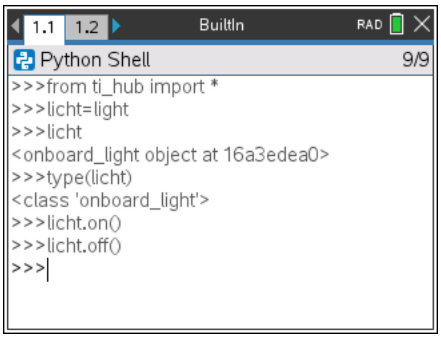
```
from ti_hub import *
ladder=[262,294,330,349,392,440,494,523]
for i in ladder:
    sound.tone(i,1)
    sleep(0.5)
```

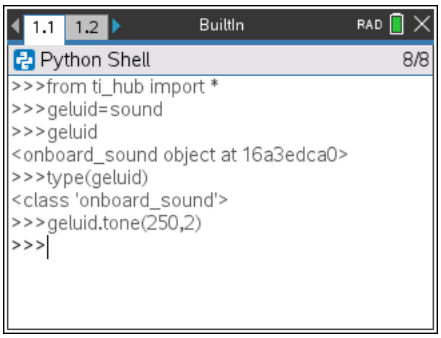


2.3. TI-Innovator Hub interne objecten

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:

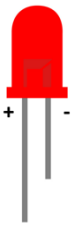






Voor het aansturen van externe apparaten, is het noodzakelijk dat eerst de specifieke objecten gedefinieerd worden.

2.4. Externe LED's

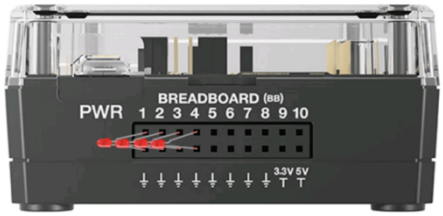


Veronderstel dat we 4 LED's verbinden met de breadboard-poorten BB1 tot en met BB 4. Verbind de lange uiteindes (Anode +) van de LED's in 1 tot en met vier en de korte (Kathode –) in de aarding poorten.

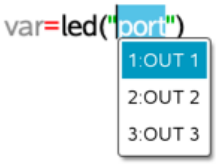
Na het selecteren van de optie 1:LED in het Add Output Device verschijnt de syntax voor het definiëren van een led-object: var=led("port").

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")
```

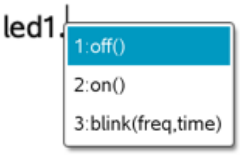


Merk op dat met de cursor in het "port"-veld, automatisch de output-poorten OUT 1, OUT 2 en OUT 3 verschijnen. Deze poorten zijn nodig bij het gebruik van Grove LED-modules.



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.** .

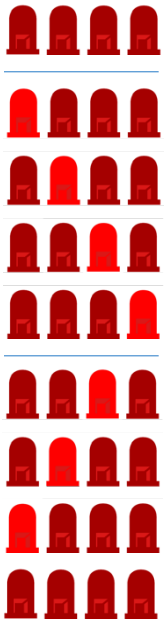


De volgende commando's zetten de vier LEDs aan:

```
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()
```

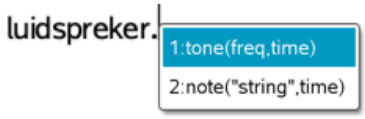


Het terug laten lopen kan met de onderstaande for-lus:

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

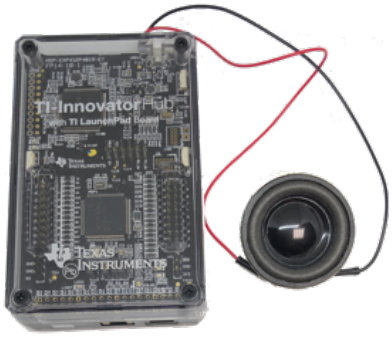
2.5. Externe Luidspreker

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:

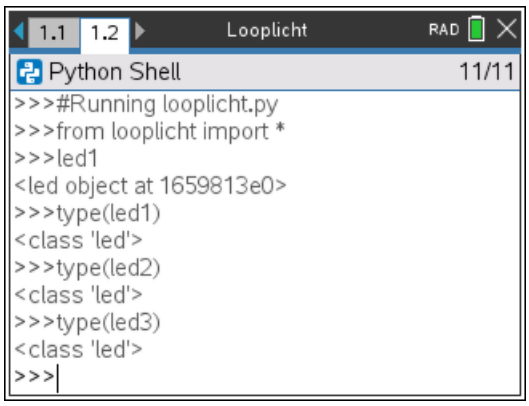
```
from ti_hub import *
luidspreker=speaker("BB 1")
for i in range(5):
    luidspreker.tone(500,1)
    sleep(1)
    luidspreker.tone(375,1)
    sleep(1)
```

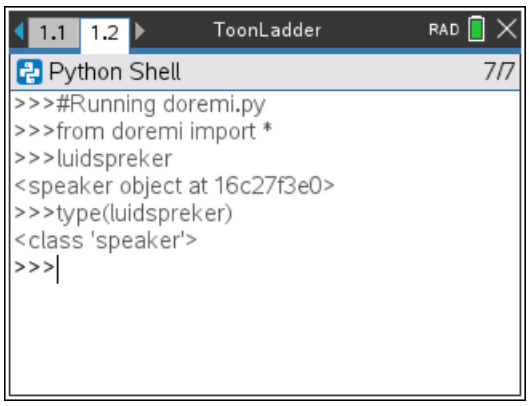
DoReMi:

```
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().

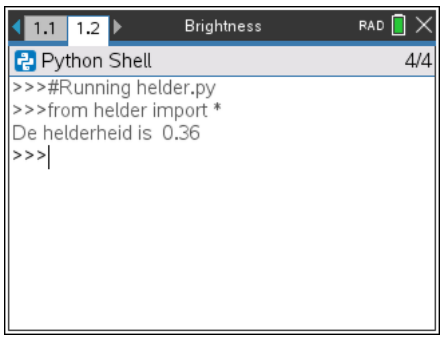
Het meten van de ingebouwde lichthelderheid-sensor gebeurt met de volgende code:

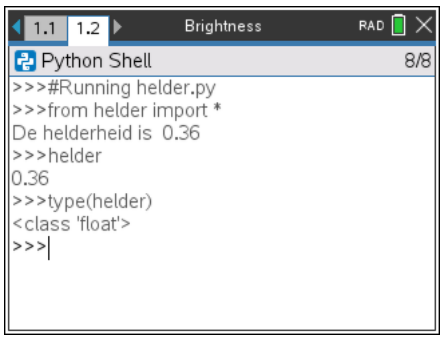
```
from ti_hub import *
brightness.measurement()
```

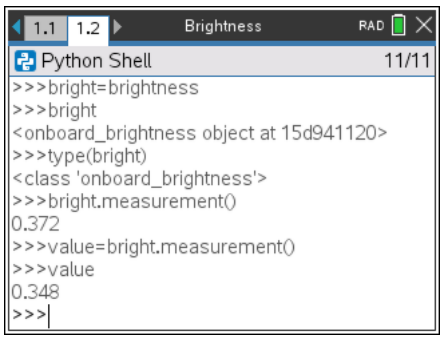
Het runnen van deze code geeft echter geen output. Het weergeven van de gemeten waarde kan als volgt:

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

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



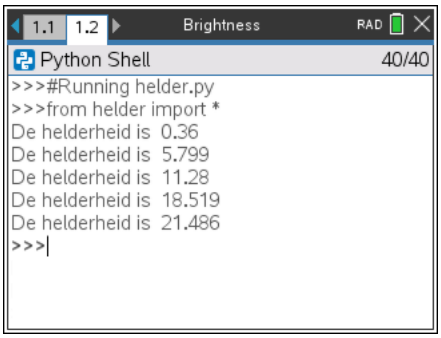




De range van de ingebouwde brightness-sensor gaat standaard van 0 tot 100.

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)
```

1. 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:

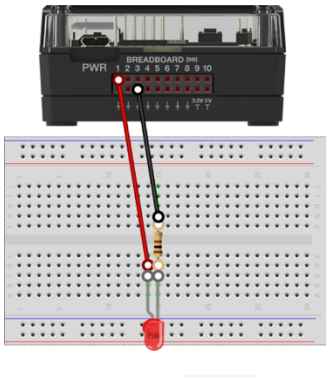
Ingebouwde RGB-led

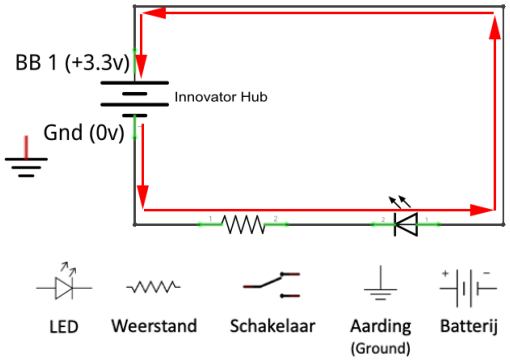
```
from ti_hub import *
color.rgb(150,150,150)
color.blink(0.5,4)
sleep(4)
color(2,4)
```

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: $V = I \cdot R$.

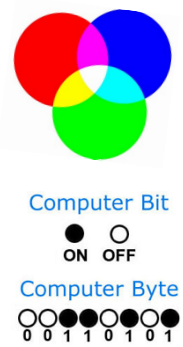




Meer van deze activiteiten op www.wil-destem.be of www.wil-destem.nl.

2. 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.

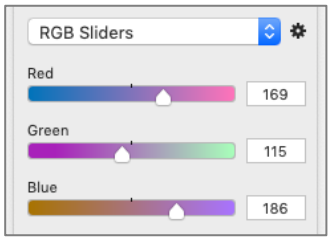


Kleuren produceren met een RGB-led komt neer op het configureren van de intensiviteit van iedere led. Omdat de led's zeer kort bij elkaar staan ziet ons oog niet de individuele kleuren maar enkel de combinatie van de kleuren.

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.

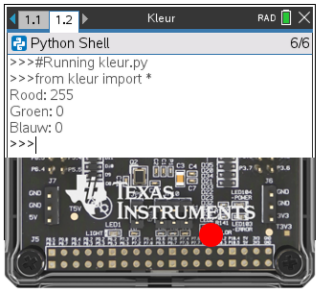


Met de onderstaande code kunnen we experimenteren met de RGB-led:

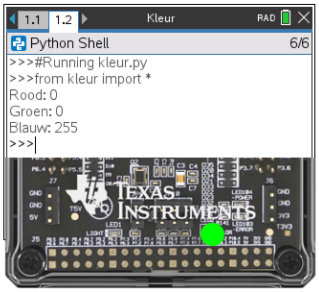
```
from ti_hub import *
r=int(input("Rood: "))
g=int(input("Groen: "))
b=int(input("Blauw: "))
color.rgb(r,g,b)
sleep(2)
color.off()
```

Hieronder enkele voorbeelden:

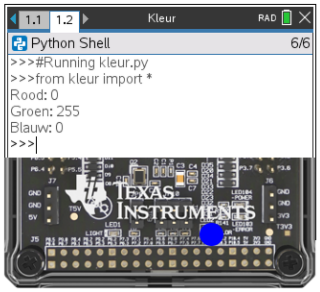
R:255 – G:0 – B:0



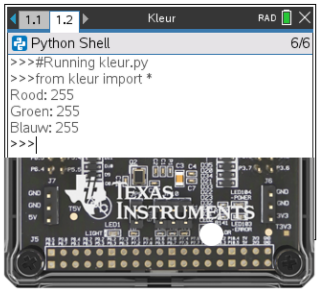
R: – G:255 – B:0



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()
```

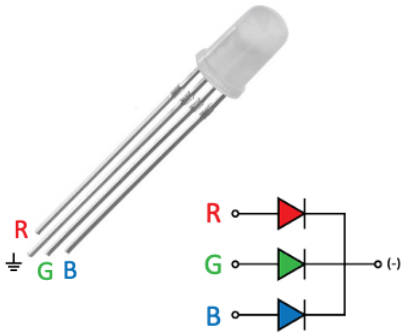
Grove RGB-LED in OUT 1

```
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()
```

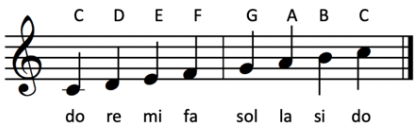
De methode set() laat het toe de intensiteit van iedere led te configureren en zo verschillende kleuren te genereren:

```
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)
```

3. 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.

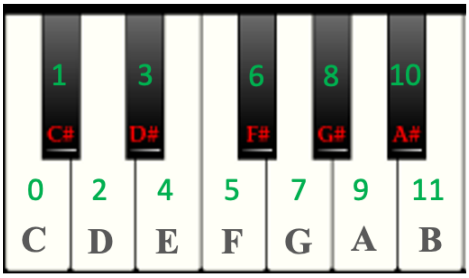
toon	1	2	3	4	5	6
C	32,7	65,4	130,8	261,6	523,2	1046
C#	34,64	69,29	138,5	277,1	554,3	1108
D	36,7	73,41	146,8	293,6	587,3	1174
D#	38,89	77,78	155,5	311,1	622,2	1244
E	41,2	82,4	164,8	329,6	659,2	1318
F	43,65	87,3	174,6	349,2	698,4	1396
F#	46,24	92,49	184,9	369,9	739,9	1479
G	48,99	97,99	195,9	391,9	783,9	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

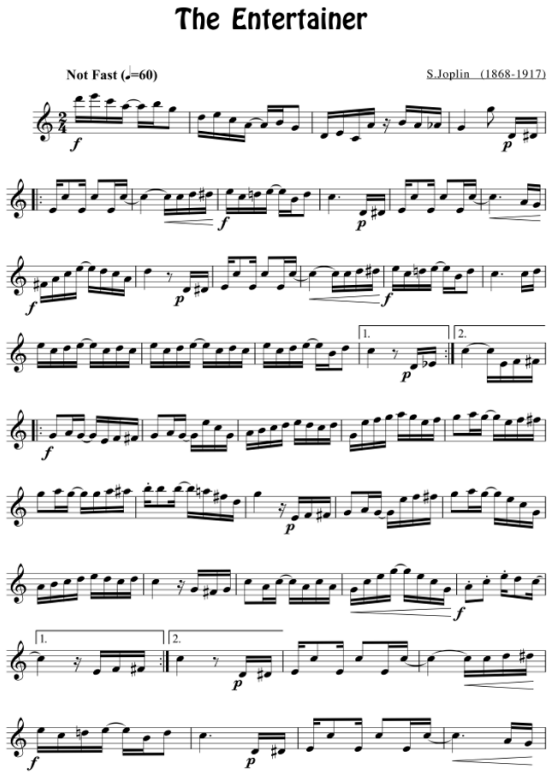
Happy Birthday	toon	tijd
1	d	0.5
2	d	0.5
3	e	1
4	d	1
5	g	1
6	f+1	2
7	d	0.5
8	d	0.5
9	e	1
10	d	1
11	a	1
12	g	2
13	d	0.5
14	d	0.5
15	d+12	1
16	b	1
17	g	1
18	f+1	1
19	e	1
20	c+12	0.5
21	c+12	0.5
22	b	1
23	g	1
24	a	1
25	g	2

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, 1, 2, 1, 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, 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, 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])
```

1. 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.

1.1. 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".

1.2. 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).

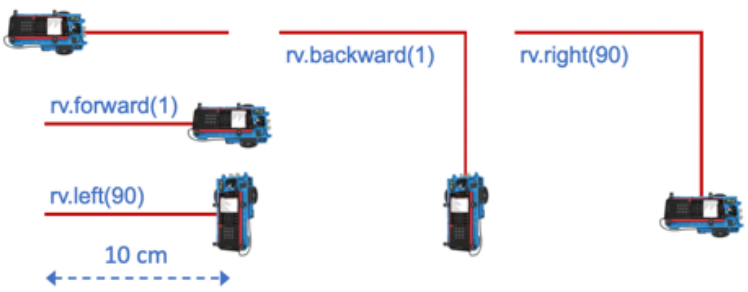
Ook aan deze commando's kan een eenheid toegevoegd worden:

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

De default-eenheid is "degrees".

1.3. Een eerste move

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



1.4. 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

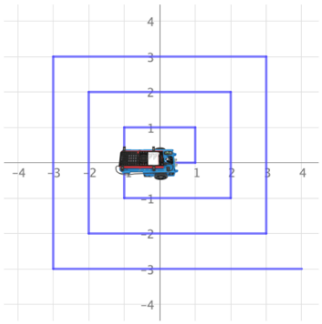
```
import ti_rover as rv

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)
```

1.5. Spiralen

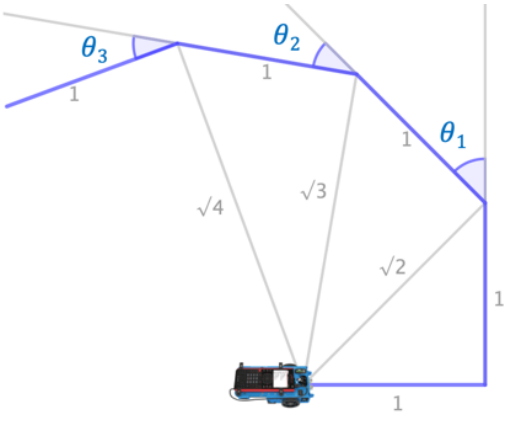
Voorbeeld 1 – Square spiral



```
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)
```



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

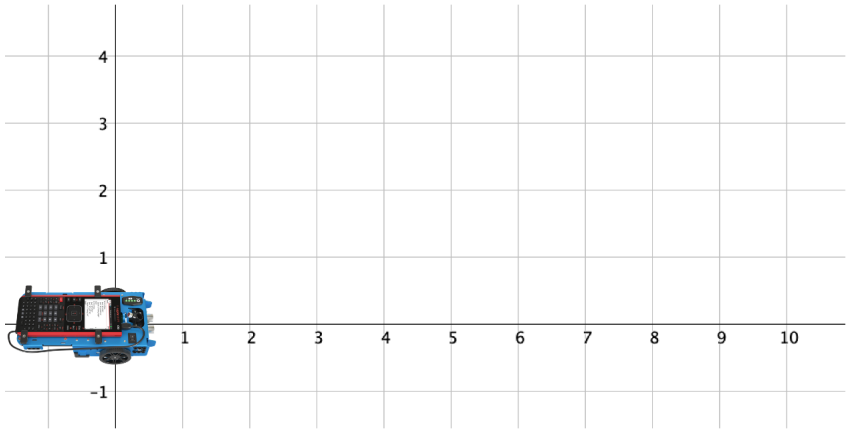
$$\theta_2 = \tan^{-1}\left(\frac{1}{\sqrt{2}}\right)$$

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

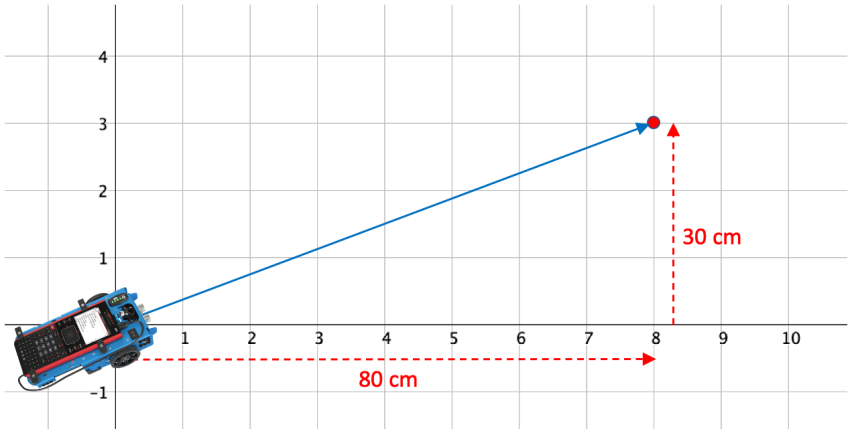
$$\vdots$$

2. Rijden via coördinaten

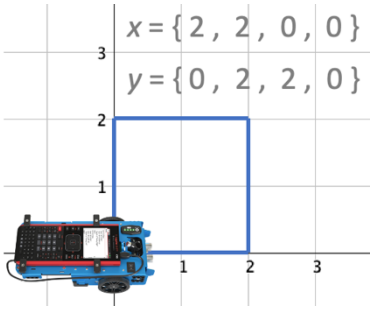
Met de code rv.to_xy(x,y) geef je de Rover de instructie naar het punt met coördinaten (x,y) te rijden in het virtuele grid met de Rover als volgt startend in de oorsprong:



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



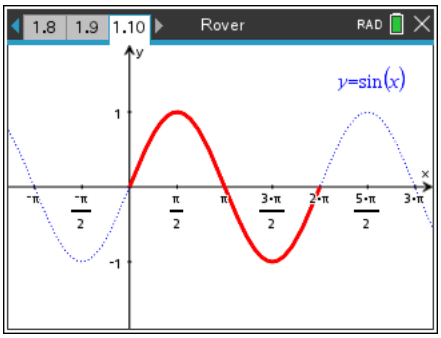
M.b.v. een for-lus rijden we het volgende vierkant:




```
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])
```

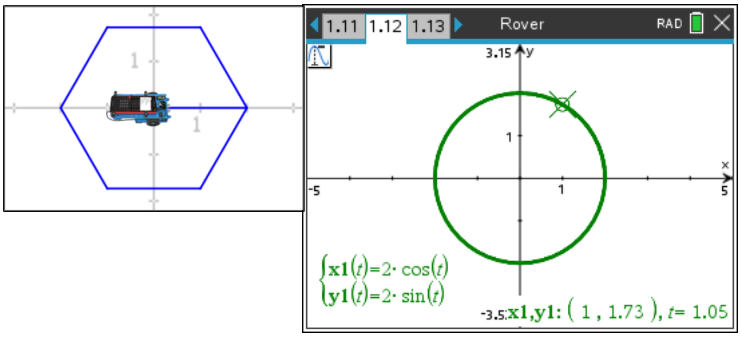
3. Het rijden van functies

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

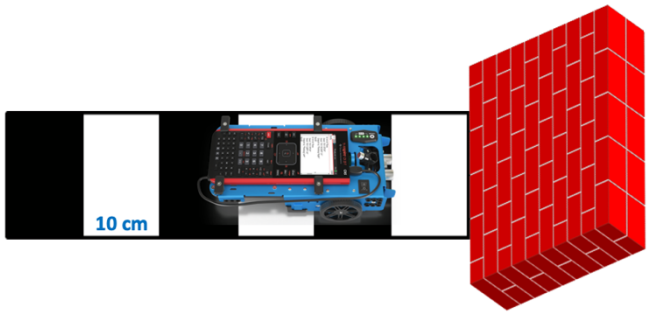



```
from math import *
import ti_rover as rv

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)
```

4. Metend rijden

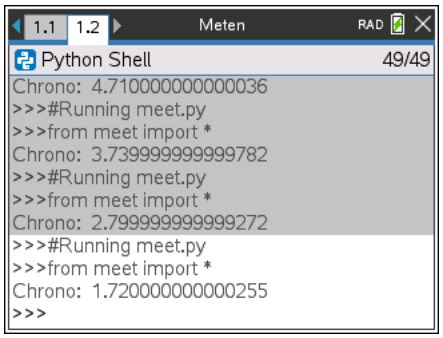
Aan de voorkant van de Rover is een ranger bevestigd waarmee de afstand (in meter) van de Rover tot een object gemeten wordt. Indien de rover te dicht komt met b.v. een muur brengt de methode stop() de Rover tot stilstand.

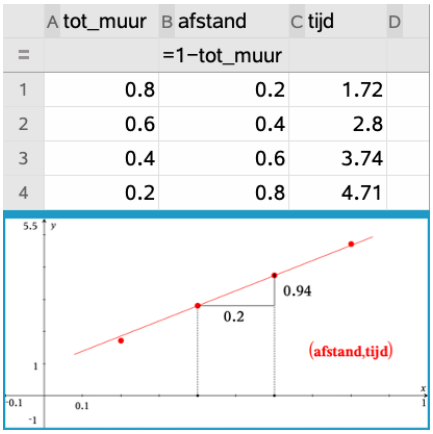
```
import ti_rover as rv
afstand=rv.ranger_measurement()
rv.forward(100)
while afstand > 0.1:
    afstand=rv.ranger_measurement()
rv.stop()
```

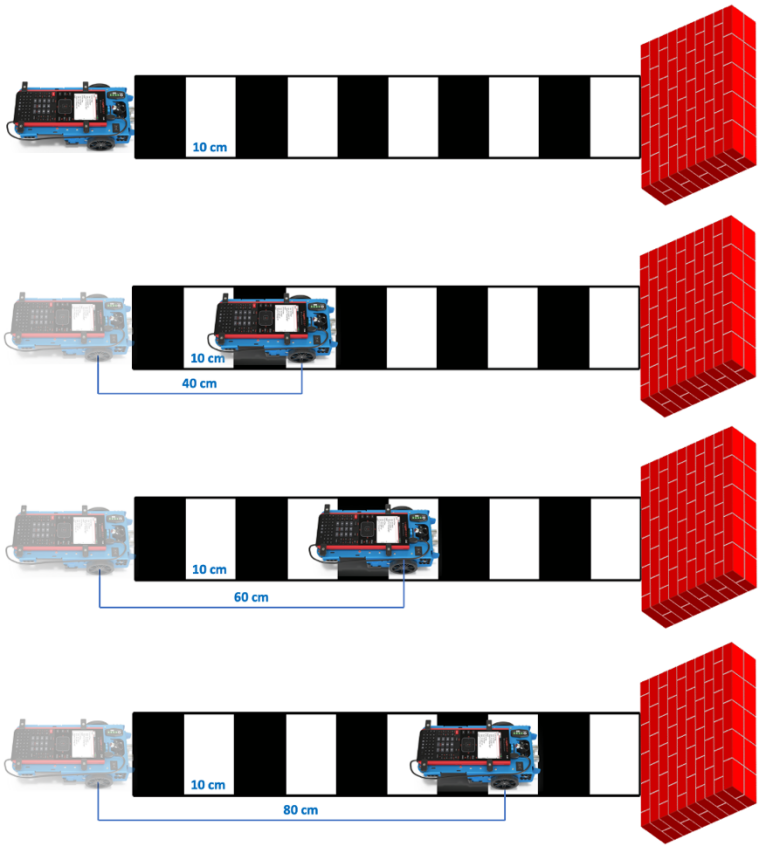


Het volgende programma berekent een ruwe benadering van de snelheid van de eenparig rechtlijnige beweging van de rover. Het statement clock() van de time-module geeft de processor tijd als een float-getal, uitgedrukt in seconden.

```
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)
```







2.80 seconden

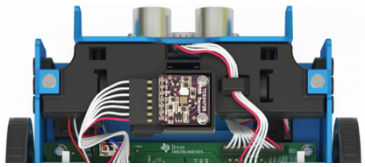
3.74 seconden

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 varieëeren 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.



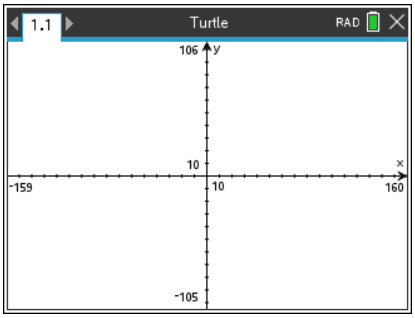
5. Rover-Turtle

Leraren van het ww T³-netwerk – Teaceher Teaching with Technology – schreven een Turtle-achtige module. Deze module kan gedownload worden via www.wil-depython.be of www.wil-depython.nl bij BootCamp Deel 3 – Programmeren van een microcontroller.

Deze module simuleert o.a. de Rover voor de voorgaande TI Python programma's, uitgezonderd deze i.v.m meten. Plaats het tns-document met de turtle-code in de PyLib-folder en Refresh Libraries.

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

De module cx_turtle maakt gebruik van de volgende scherminstellingen: (xim,xmax) = (-159,160) en (ymin,ymax) = (-105,106).



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

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

Bij start van een programma bevindt rv zich in de oorsprong. De functie rv.home() brengt rv steeds terug naar de oorsprong en dit zonder tekenen.

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.color(r,g,b) en rv.pensize(value 0, 1 of 2) brengen wat kleur en variatie in het tekenen.

Enkele programma's

TI-Innovator Rover

Vierkant

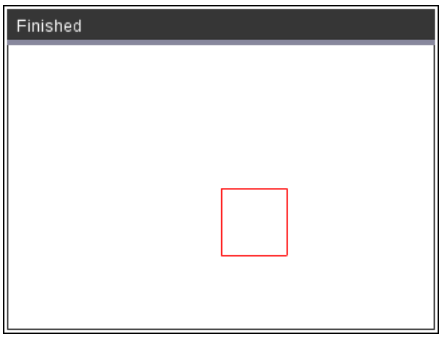
```
import ti_rover as rv
  for i in range(4):
    rv.forward(3)
    rv.right(90)
```

Driehoek

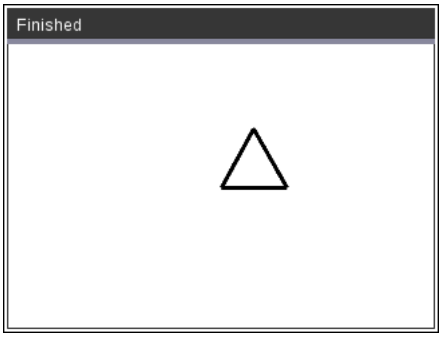
```
import ti_rover as rv
  for i in range(3):
    rv.forward(3)
    rv.left(120)
```

Turtle Rover

```
from cx_turtle import *
rv=turtle()
rv.color(255,0,0)
for i in range(4):
    rv.forward(50)
    rv.right(90)
```



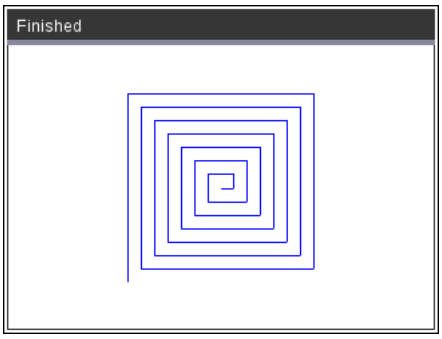

```
from cx_turtle import *
rv=turtle()
rv.pensize(1)
for i in range(3):
    rv.forward(30)
    rv.left(120)
```



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)
```

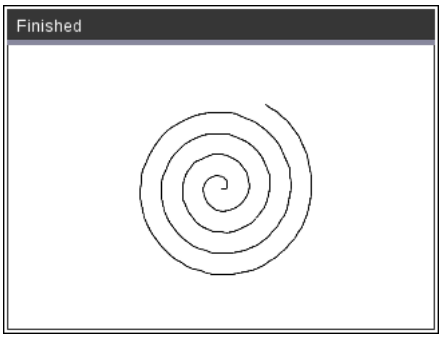
```
from cx_turtle import *
rv=turtle()
rv.color(0,0,255)
for i in range(1,15):
    rv.forward(i*10)
    rv.left(90)
    rv.forward(i*10)
    rv.left(90)
```



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)
```

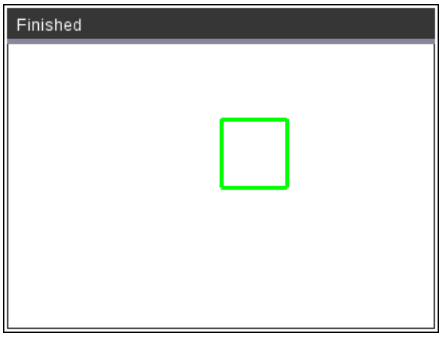
```
from cx_turtle import *
from math import *
rv=turtle()
rv.forward(5)
rv.left(90)
rv.forward(5)
for i in range(1,200):
    h1=1/sqrt(i)
    h2=atan(h1)*180/pi
    rv.left(h2)
    rv.forward(5)
```



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])
```

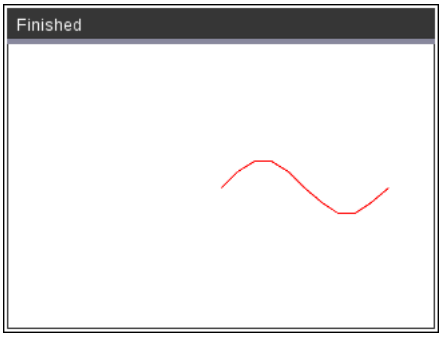
```
from cx_turtle import *
rv=turtle()
rv.color(0,255,0)
rv.pensize(1)
x=[50,50,0,0]
y=[0,50,50,0]
for i in range(len(x)):
    rv.goto(x[i],y[i])
```



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))
```

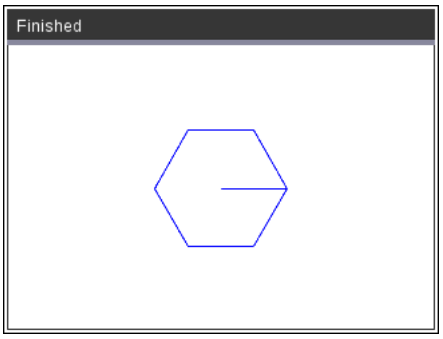
```
from cx_turtle import *
from math import *
rv=turtle()
rv.color(255,0,0)
points=10
scale=2*pi/points
x=[i*scale for i in range(points+1)]
for i in x:
    rv.goto(20*i,20*sin(i))
```



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 cx_turtle import *
from math import *

rv=turtle()
rv.color(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)
```



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.

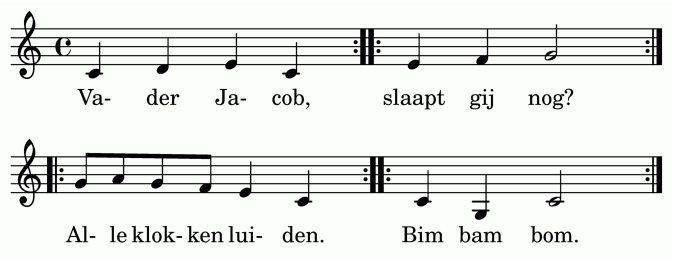
2. Straatverlichting

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.

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
C G C
Bim Bam Bom
C G C
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!

4. Rij/Wandel de grafiek

a. Cirkelbeweging

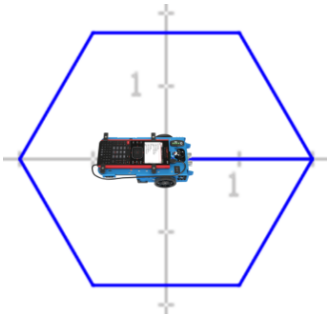
Voor de onderstaande code krijgen we de volgende resultaten.

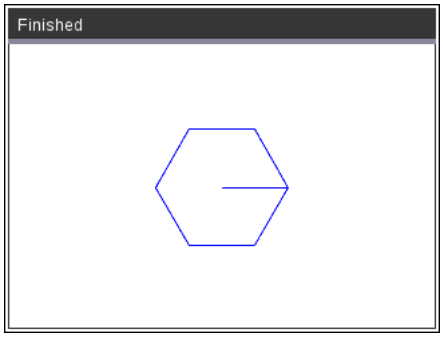
TI-Innovator Rover

```
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)
```

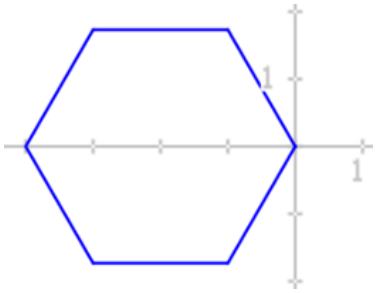
CX Turtle

```
from cx_turtle import *
from math import *

rv=turtle()
rv.color(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)
```





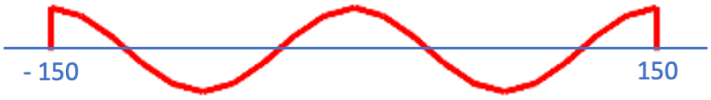
Pas de code voor de parameterkromme aan om de volgende output te krijgen.



b. SinRide

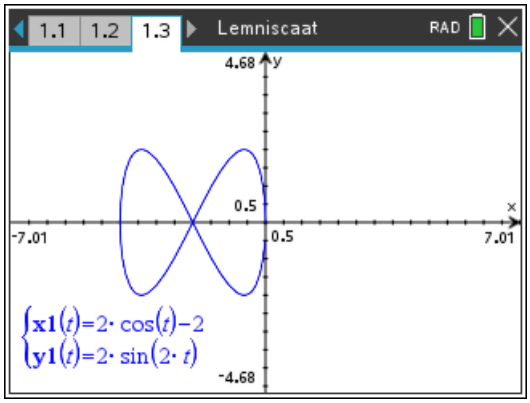
met fase-verschil van grootte $\frac{\pi}{2}$

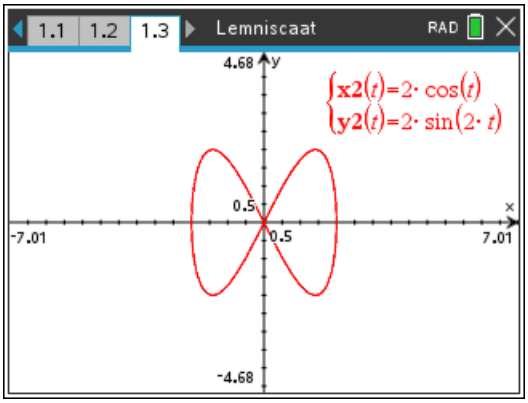
- Laat de Rover een cosinus-curve rijden/tekenen startend in de oorsprong tot 4π.
- Teken de volgende cosinus curve met de CX Turtle met amplitude 20 pixels.



c. Oneindige lus

Laat de Rover een oneindige lus rijden/tekenen:

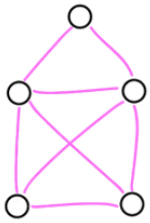




Voor de CX-Turtle geldt dat 1 eenheid gelijk is aan 1 pixel. Voor het tekenen van een oneindig lus met de Turtle best een schaalvergroting toevoegen, b.v. 30 in de x-richting en 20 in de y-richting.

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.

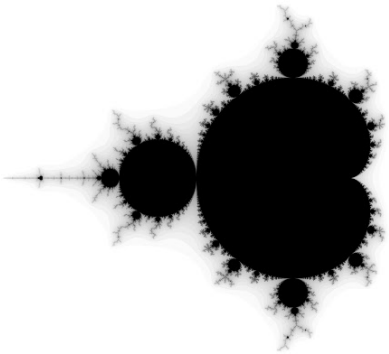


Schrijf een programma dat de Rover (of de CX-Turtle) zo'n huisje laat tekenen:

- 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, $a \in \mathbb{R}$:

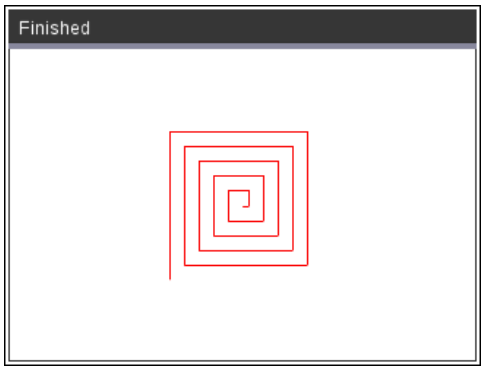
- Parametervoorstelling
  $x(t) = 2a \cdot \cos(t)\,(1 - \cos(t))$
  $y(t) = 2a \cdot \sin(t)\,(1 - \cos(t))$
- Polaire coördinaten
  $r(\theta) = 2a(1 - \cos(\theta))$

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).

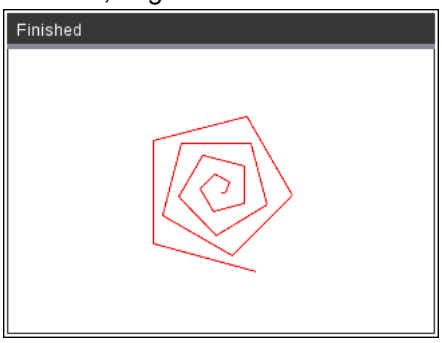
5. Spiraal-plezier

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

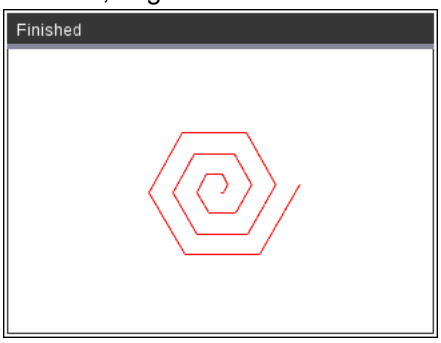
```
from cx_turtle import *
rv=turtle()

width=5
angle=90
segments=20

rv.color(250,0,0)

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



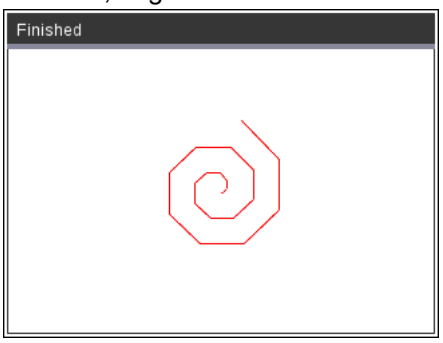
width=4 ; angle=75



width=3 ; angle=60

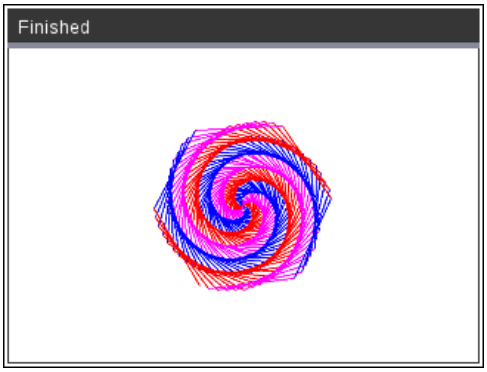


width=2 ; angle=45

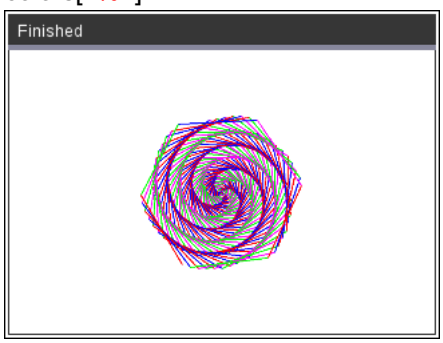


Experimenteren met kleur kan als volgt:

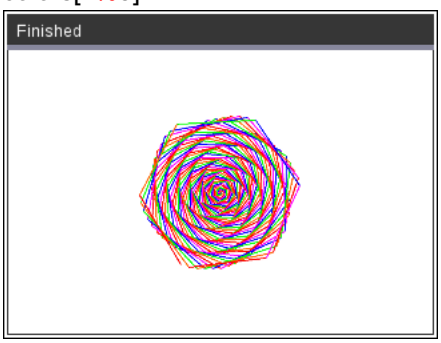
```
from cx_turtle import *
rv=turtle()

width=6
angle=59
segments=360

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

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



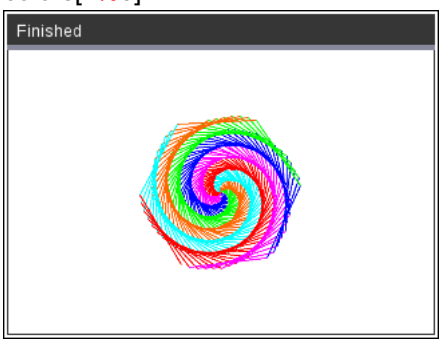
colors[x%4]



colors[x%5]

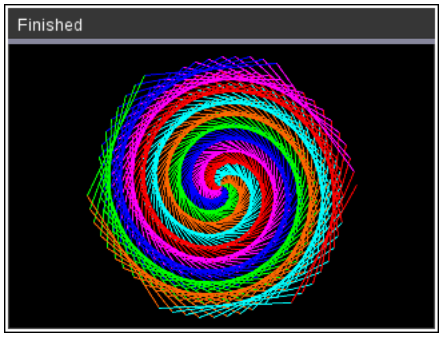


colors[x%6]

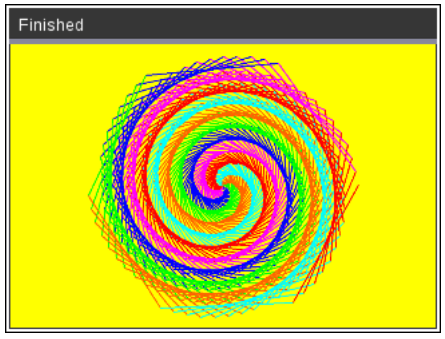


Een manier, om met deze relatief beperkte Turtle-module, de achtergrond te kleuren is gebruik te maken van de dot()-functie(methode) en zo b.v. een gevulde cirkel tekenen met middelpunt de oorsprong en straal 200:

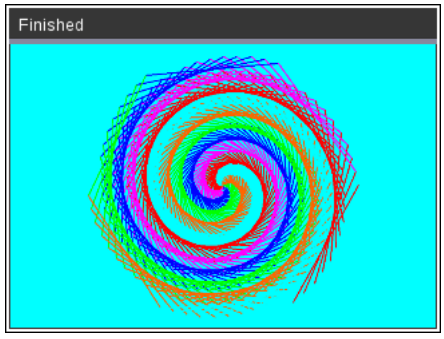
```
rv.home()
rv.color('black')
rv.dot(200)
```



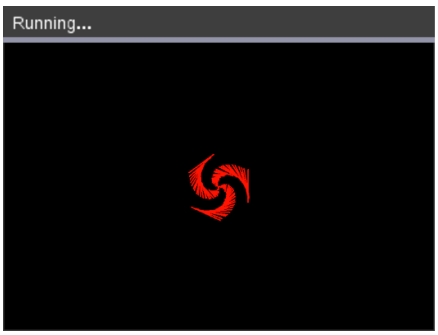

```
rv.home()
rv.color('yellow')
rv.dot(200)
```

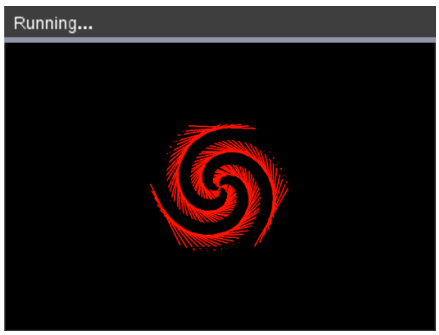


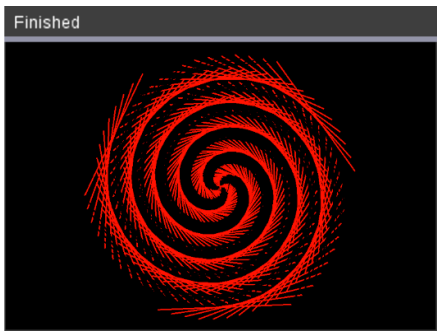

```
r.home(()
rv.color('cyan')
rv.dot(200)
```



Veel plezier met het kunstig tekenen van spiralen!







1. 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()
```

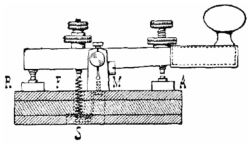
De boodschap die in het dashboard wordt weergegeven moet als een string ingegeven worden. Met de stringmethode format() kunnen meetwaarden toegevoegd worden.

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()
```

2. 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:

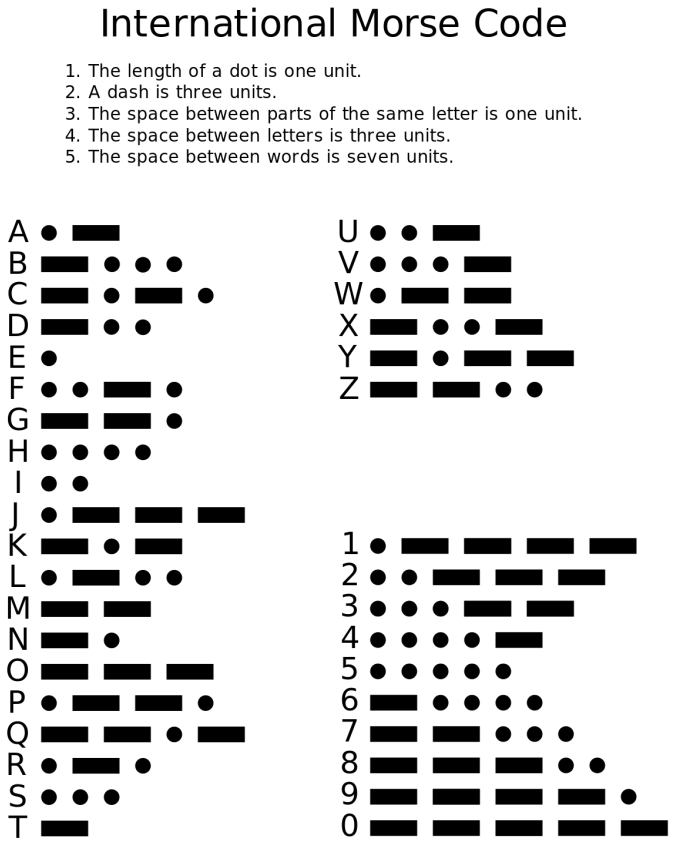
- sleutel naar beneden (stroom) of naar boven (geen stroom)
- tijdsduur kort of lang

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

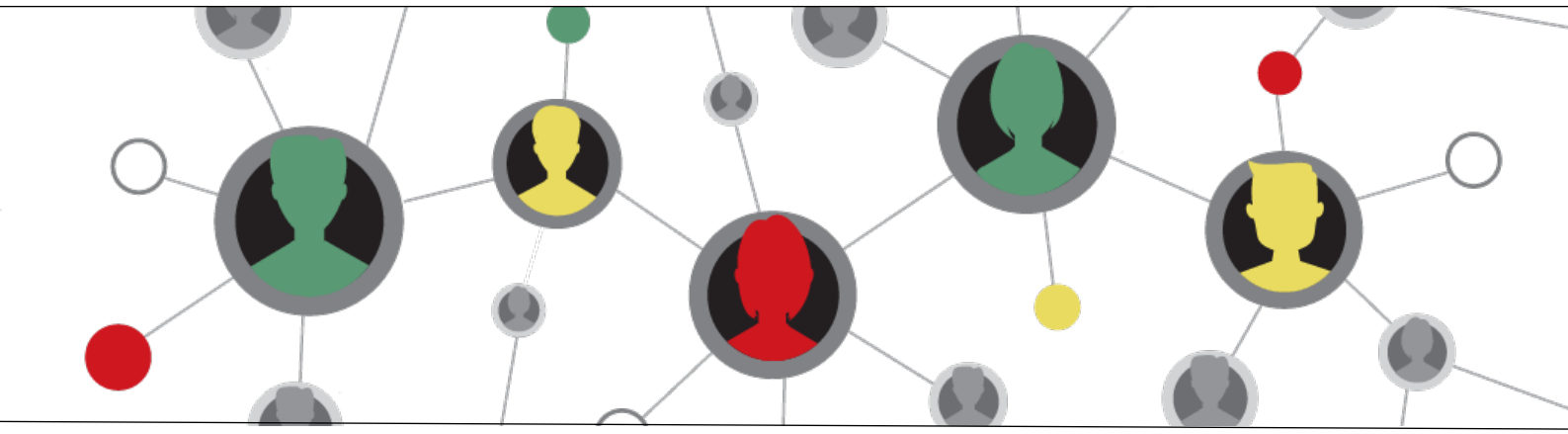
```
alfabet={
    "A": "·-",
    "B": "-···",
    "C": "-·-·",
    "D": "-··",
    "E": "·",
    "F": "··-·",
    "G": "--·",
    "H": "····",
    "I": "··",
    "J": "·---",
    "K": "-·-",
    "L": "·-··",
    "M": "--",
    "N": "-·",
    "O": "---",
    "P": "·--·",
    "Q": "--·-",
    "R": "·-·",
    "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)
```



T³ NEDERLAND T³ VLAANDEREN

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

Texas Instruments