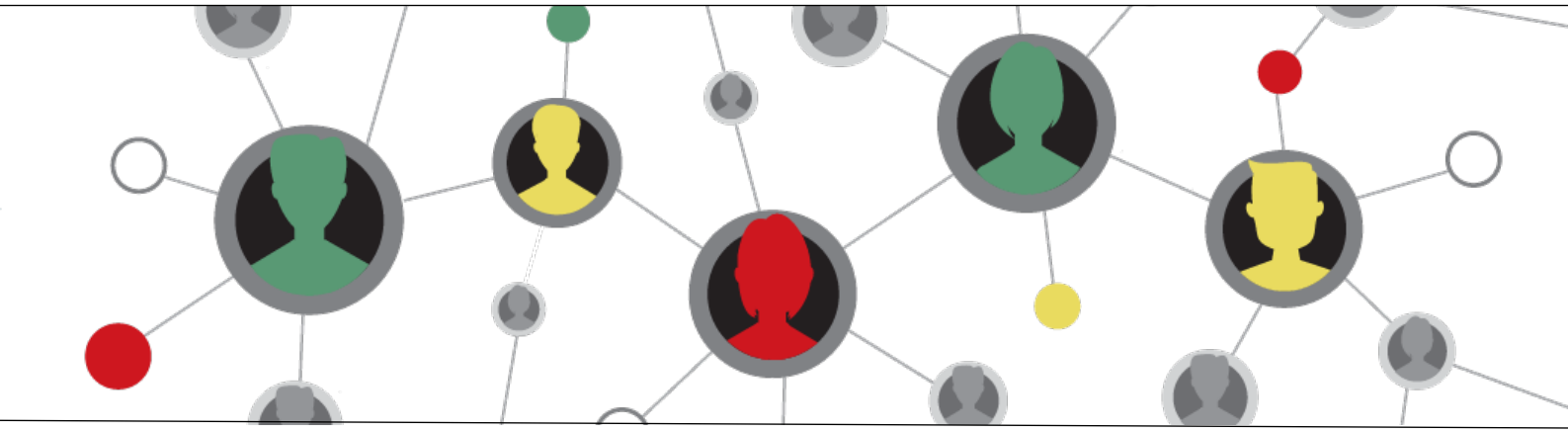
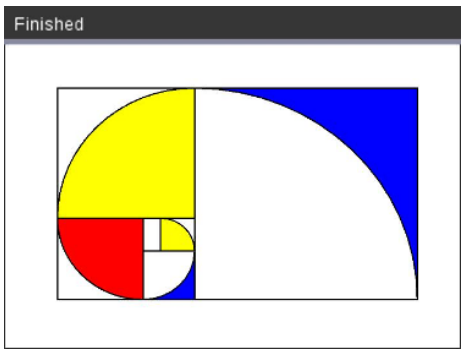
TI Python BootCamp

Deel 4 – Data-analyse & grafisch programmeren





T³ NEDERLAND

Teachers Teaching with Technology™

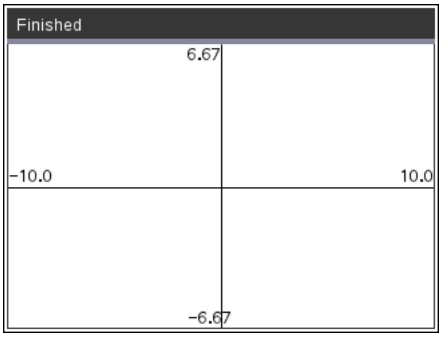
T³ VLAANDEREN

1. 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")
```

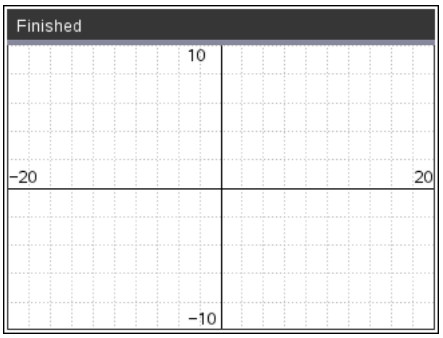


De volgende opties zijn voor axes() beschikbaar:

- "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")
```

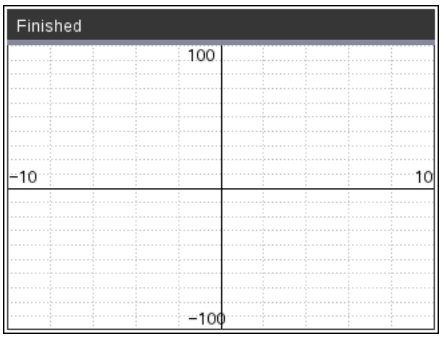


De opties voor de grid-stijl zijn:

- "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)
```

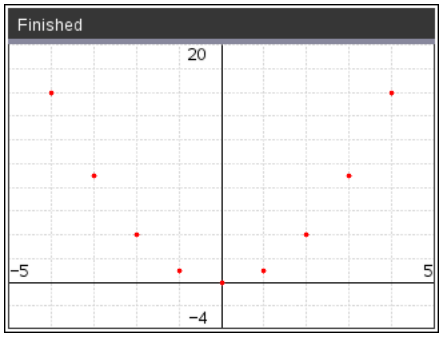


2. Grafieken

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.

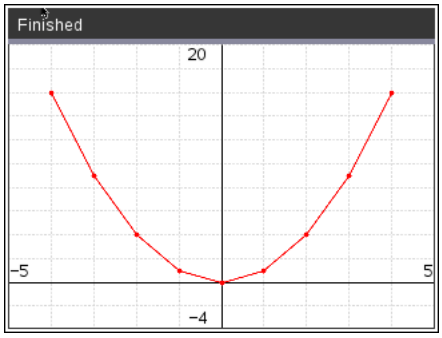
Een voorbeeld voor $f(x) = x^2$.

```
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")
```



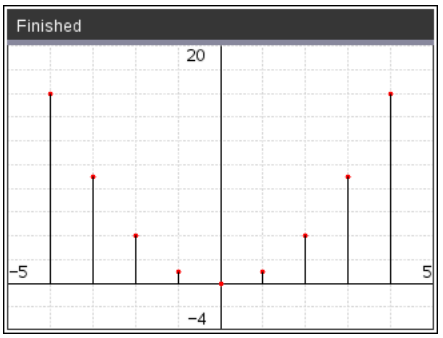
Toevoegen van de onderstaande code verbindt de punten van de grafiek door een lijnstuk:

```
    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"
- "."

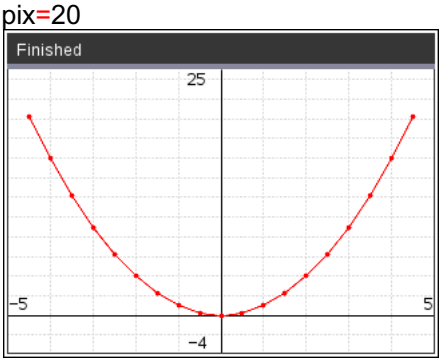
En voor pen("size","style"):

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

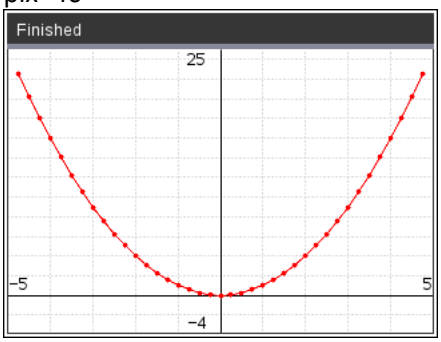
Het verfijnen van de te plotten grafiek kan als volgt:

```
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))
```

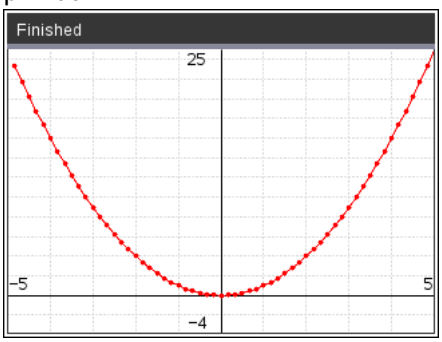
pix=20



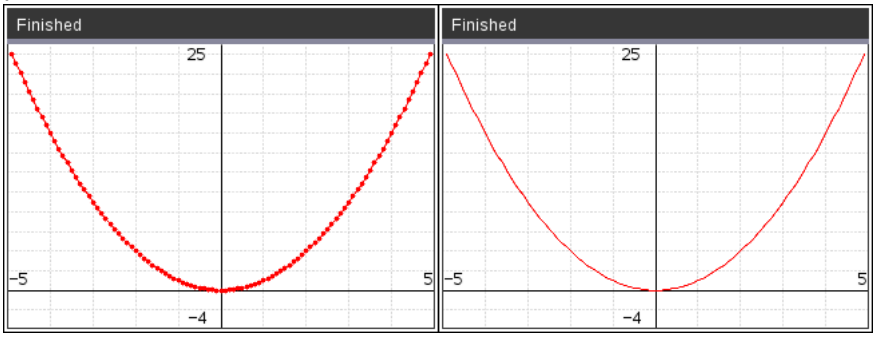
pix=40



pix=60

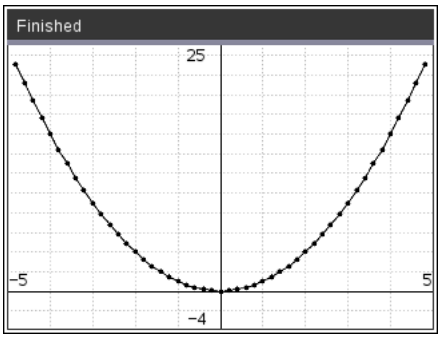


pix=100



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")
```



3. Plotten van data

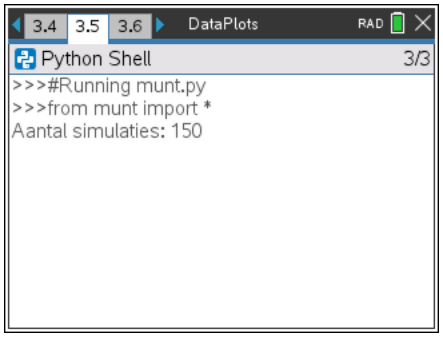
3.1. Kans-simulatie

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")
```

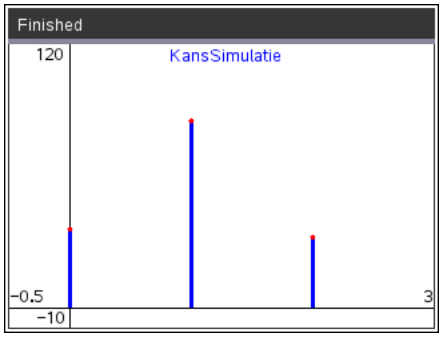


Definitie mogelijke uitkomsten en input aantal herhalingen experiment:

```
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
```

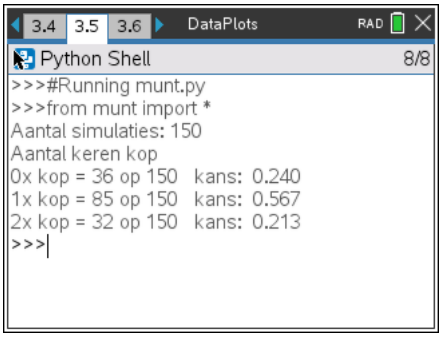


Definitie plot-venster en plotten van een titel:

```
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")
```

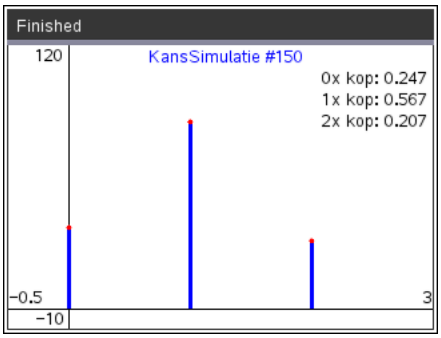


Weergeven van de benaderingen van de kans in de shell.

```
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))
```

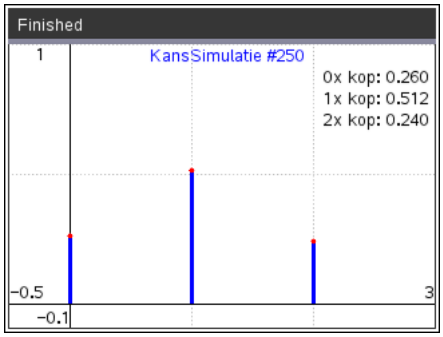
De kansbenaderingen kunnen ook als volgt weergegeven worden in het plot-venster:

```
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")
```

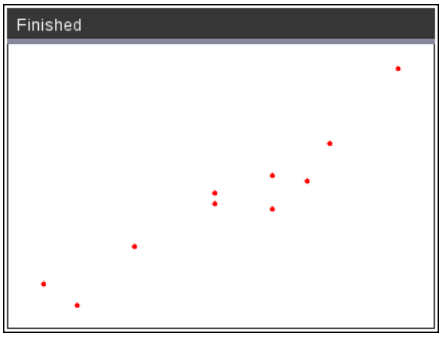


3.2. Puntenwolk

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

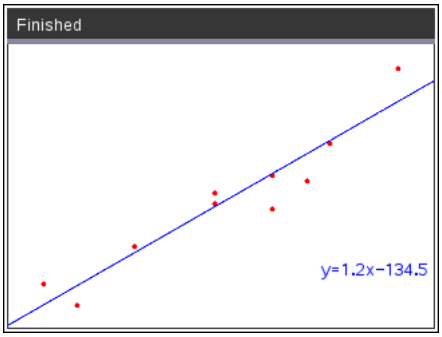
```
import ti_plotlib as plt

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
1.215261958997722
>>>plt.b
-134.4861047835991
```

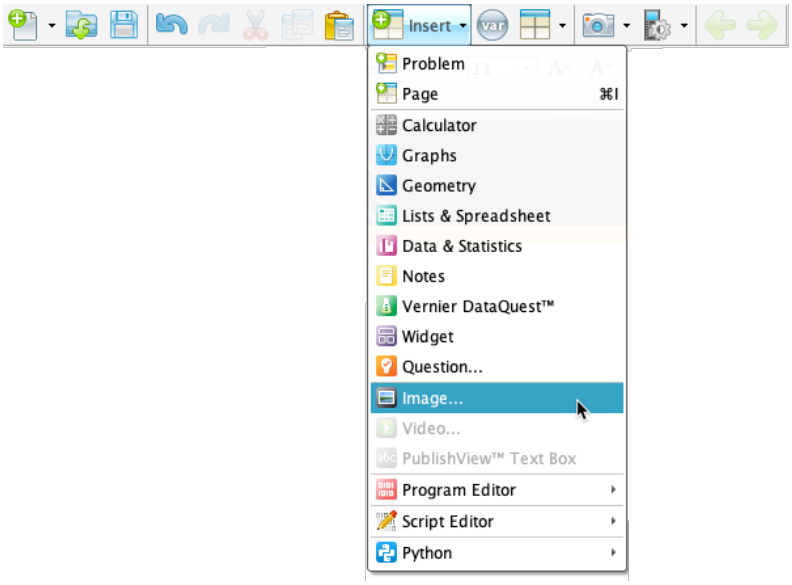
1. Digitale afbeeldingen

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.

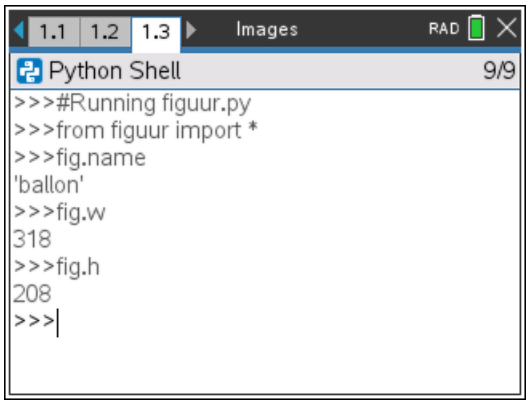


Het oproepen van de figuur ballon in een Python-programma gebeurt met deze code:

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

Voor een image-object zijn de volgende methodes beschikbaar:

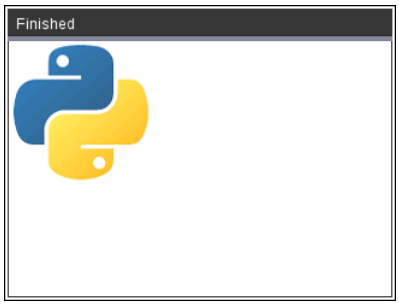
- get_pixel() leest de rgb-waarde van de pixel op posite (x,y),
- set_pixel(x,y,rgb-tuple) geeft de pixel op positie (x,y) de rgb-waarde bepaat door de tuple,
- show_image(x,y) toont de figuur met de linkerbovenhoek op positie (x,y),
- w, h, en name geeft respectievelijk de breedte, de hoogte of naam.



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:

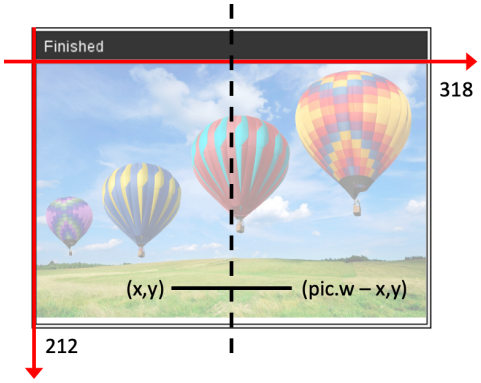
$$\text{show\_image}\left(\frac{318-pyth.w}{2}, \frac{212-pyth.h}{2}\right)$$

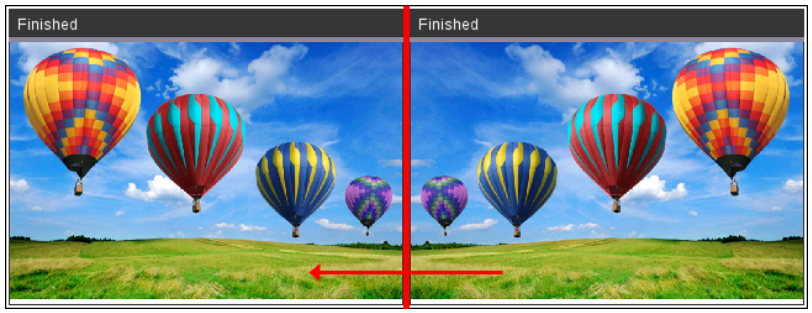


2. Transformaties

2.1. 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)
```

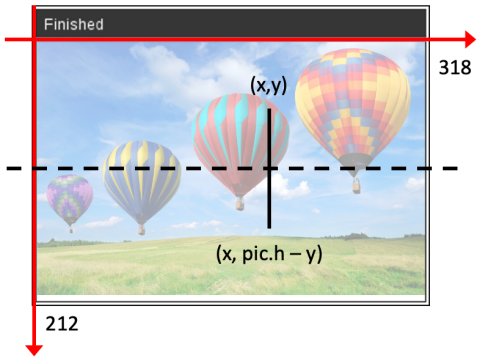
Merk op dat in de code gebruik gemaakt wordt van (pic.w – 1).

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.

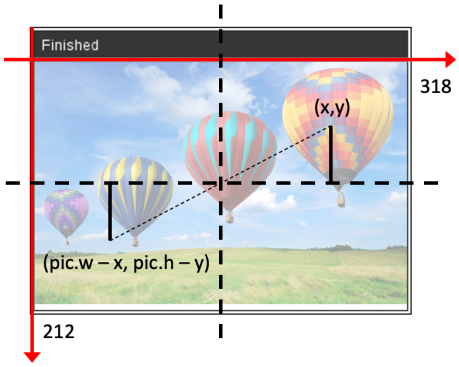
Indien we dit statement uit de lus halen, wordt alles eerst berekent en dan getoond.

2.2. Flip Verticaal



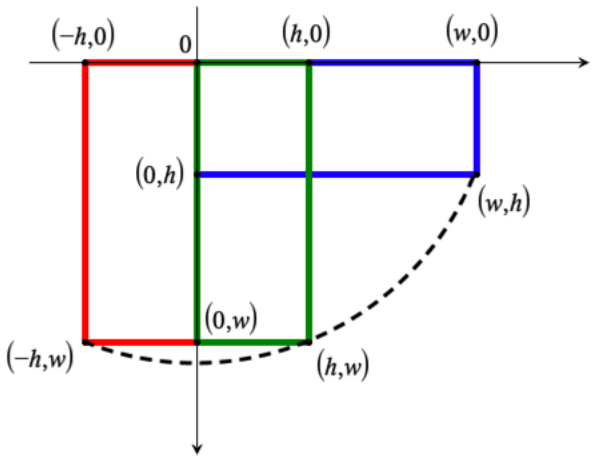

```
from ti_image import *

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)
```

2.3. 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)
```

2.4. Roteer 90° Rechts



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

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

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):

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

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.

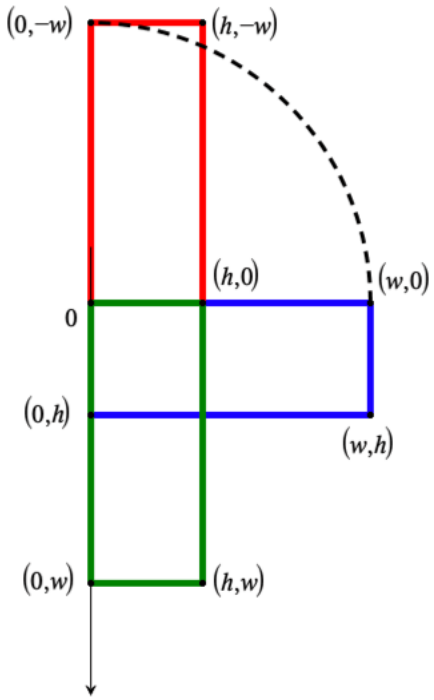
```
from ti_image import *
from ti_draw import *
pic=load_image("ballon")
pic.show_image(0,0)
```


```
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)
```

2.5. 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:

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

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):

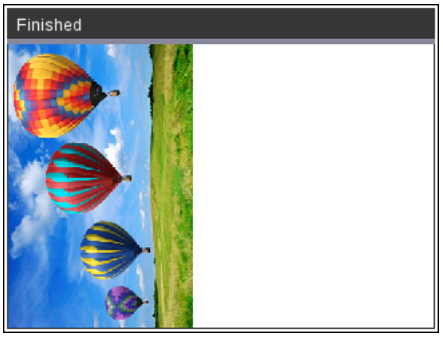
$$G: (x, y) \longmapsto (y, w - x)$$

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)
```



3. 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.

3.1. 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

3.2. 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)
```

3.3. 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

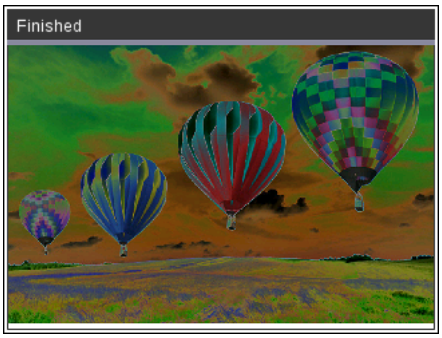
```
from ti_image import *
pic=load_image("ballon")
pic.show_image(0,0)
for x in range(0,pic.w):
    for y in range(0,pic.h):
        rgb=pic.get_pixel(x,y)
        r=255-rgb[0]
        g=255-rgb[1]
        b=255-rgb[2]
        inv=(r,g,b)
        pic.set_pixel(x,y,inv)
    pic.show_image(0,0)
```

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)



3.4. 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

3.5. 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

1. 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:

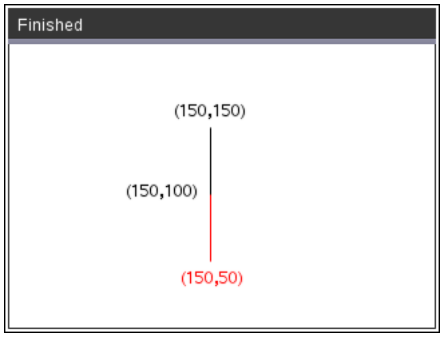
```
from ti_draw import *
dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])
draw_text(2,2,"(0,0)")
draw_text(262,195,"(318,212)")
```

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

1.1. Segmenten

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

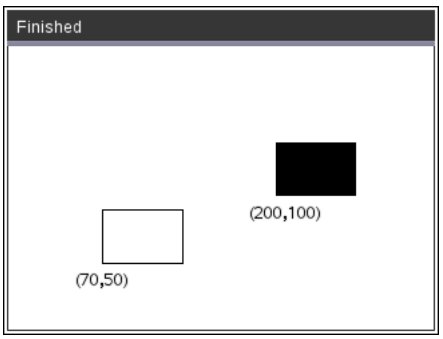
```
from ti_draw import *
dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])
draw_line(150,100,150,150)
draw_text(87,95,"(150,100)")
draw_text(123,155,"(150,150)")
set_color(255,0,0)
draw_line(150,100,150,50)
draw_text(128,30,"(150,50)")
```



1.2. Rechthoeken

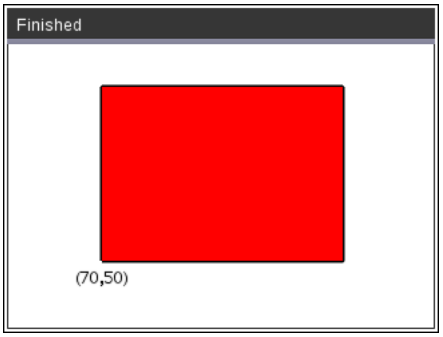
draw_rect(x,y,breedte,hoogte) – fill_rect(x,y,breedte,hoogte) tekent een rechthoek zoals hieronder aangegeven met (x,y) als linkerbenedenhoek.

```
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.

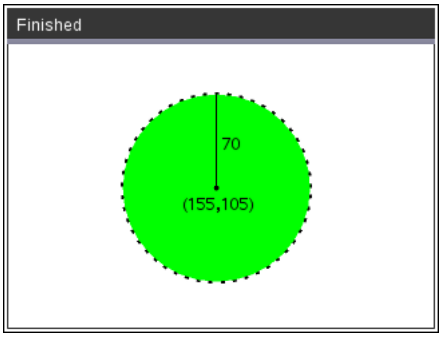
```
from ti_draw import *
dim=get_screen_dim()
set_window(0,dim[0],0,dim[1])
draw_text(50,30,"(70,50)")
xcoord=[70,250,250,70,70]
ycoord=[50,50,180,180,50]
set_pen("medium","solid")
draw_poly(xcoord,ycoord)
set_color(255,0,0)
fill_poly(xcoord,ycoord)
```



1.3. 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)
```

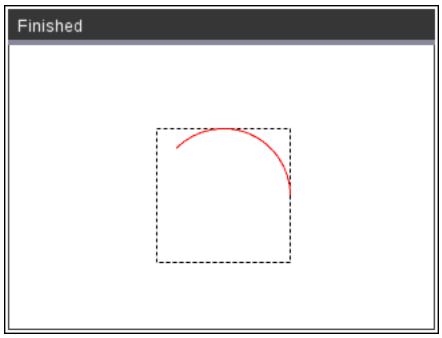


1.4. Cirkelbogen

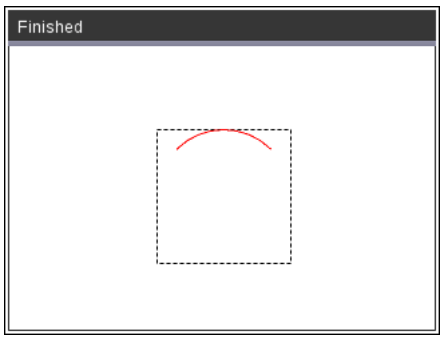
draw_arc(x,y,breedte,hoogte,beginhoek,hoekgrootte) – fill_arc(x,y,breedte,hoogte,beginhoek,hoekgrootte)

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.

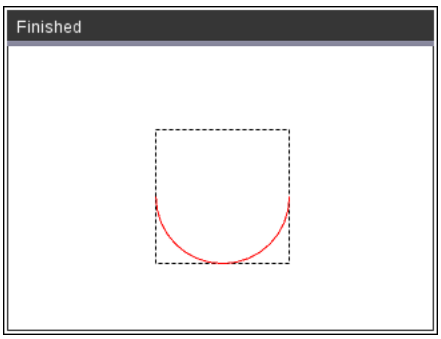
```
from ti_draw import *

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)
```



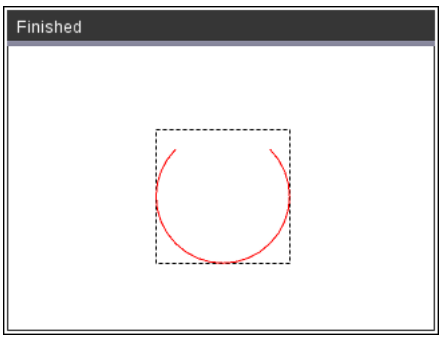
Nog enkele voorbeelden



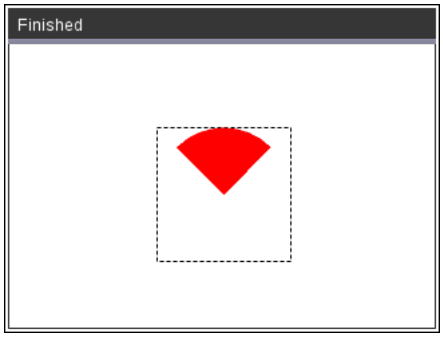
draw_arc(110,50,100,100,45,90)



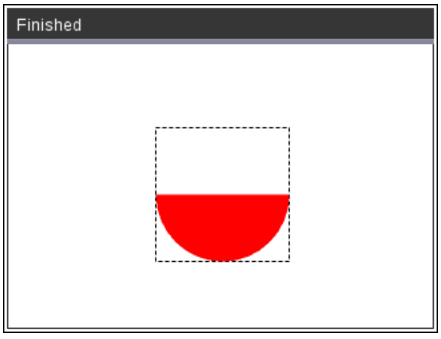
draw_arc(110,50,100,100,180,180)



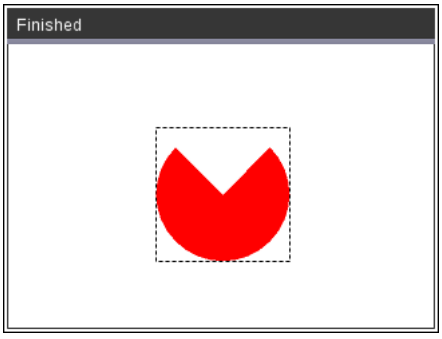
draw_arc(110,50,100,100,135,270)



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



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



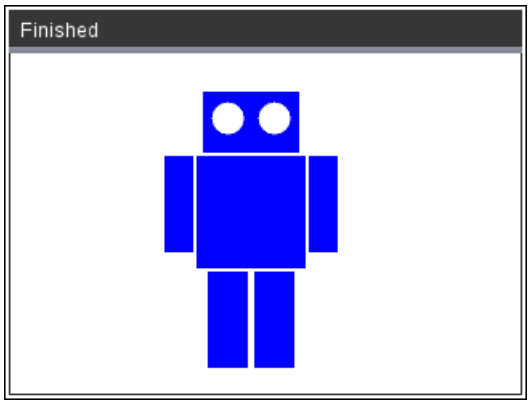
fill_arc(110,50,100,100,135,270)

2. Creatief met lijnen en bogen

2.1. Robot

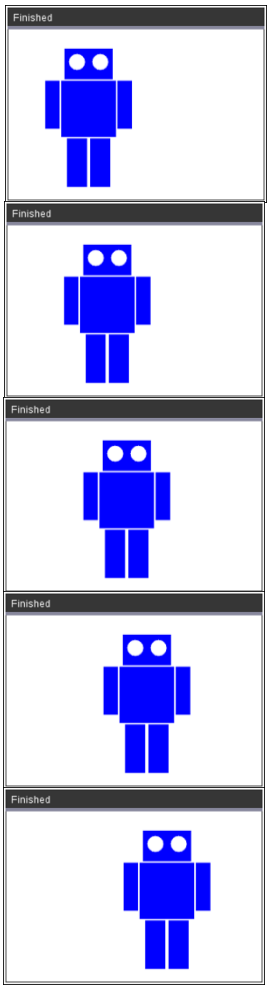
Met enkele rechthoeken en twee cirkels tekenen we onderstaande robot:

```
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.

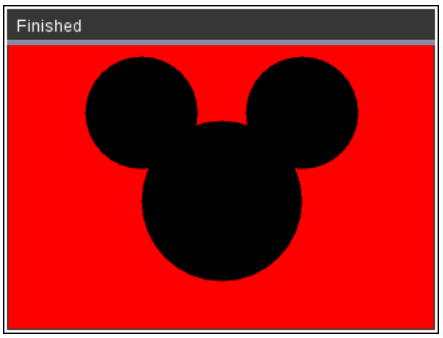
use_buffer() zorgtdat alles in de achtergrond(geheugen) wordt getekend totdat
paint_buffer() de buffer tekent.

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).

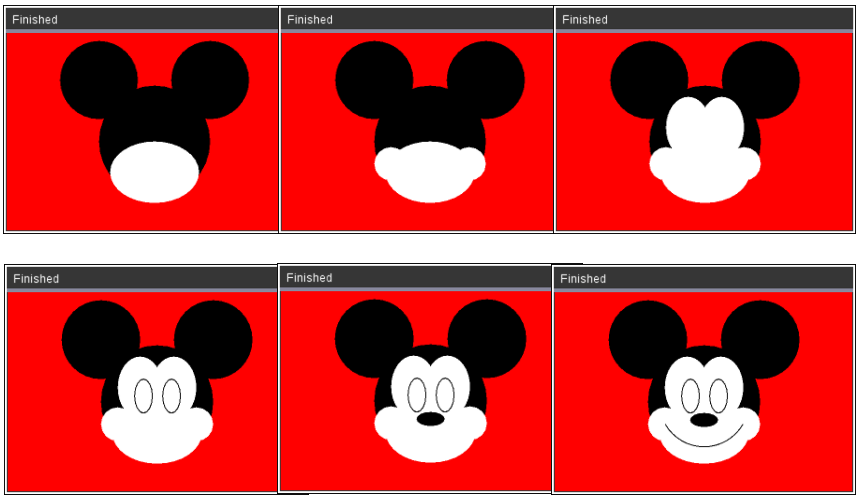
2.2. Mickey

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)
```

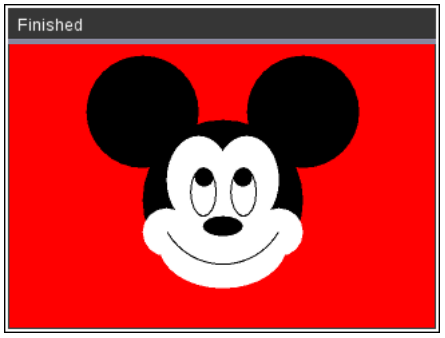



```
set_color(255,255,255)
fill_arc(-48,-76,96,66,0,360)

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)
```



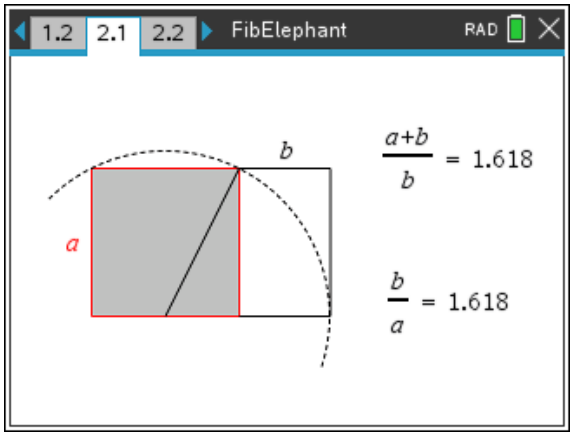
En tenslotte de pupillen van de ogen:

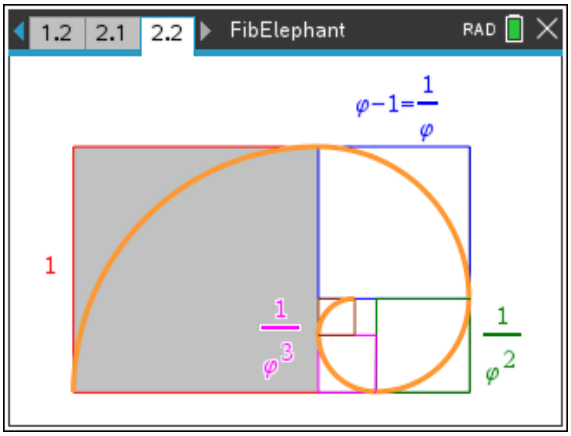
```
fill_circle(15,7.3,7)
fill_circle(-14,7.3, 7)
```



2.3. 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.

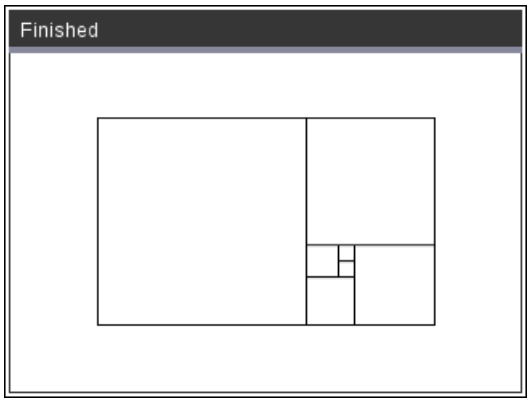


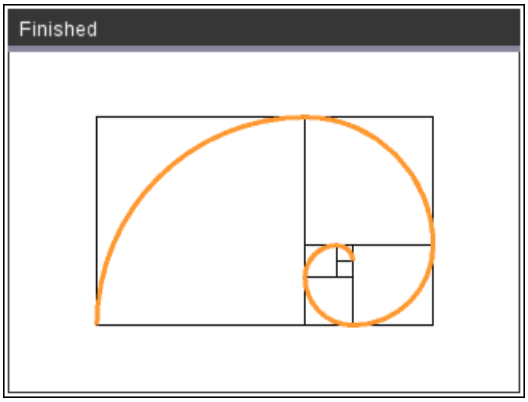


Vermits de rij met verhoudingen van opeenvolgende Fibonacci-getallen convergeert naar φ, kunnen we met de rij van Fibonacci een goudenspiraal benaderen.

De volgende code tekent de Fibonacci-spiraal:

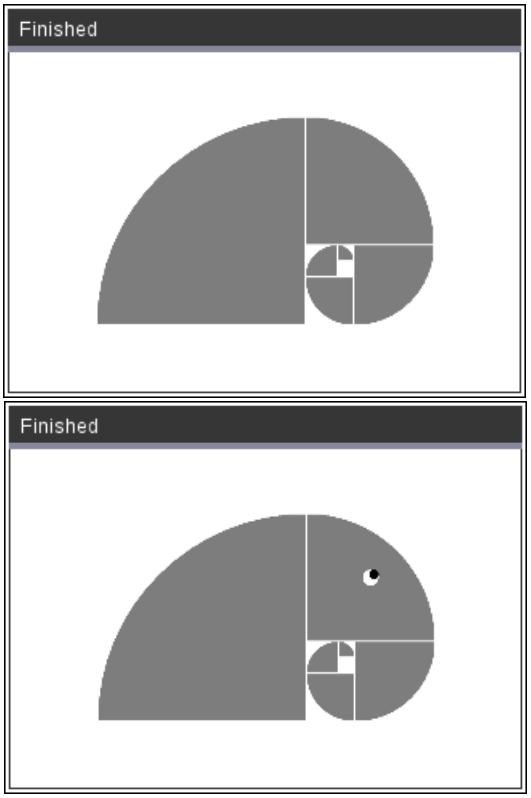
```
from ti_draw import *
dim=get_screen_dim()
set_window(-54,dim[0]-54,-42,dim[1]-42)
draw_rect(0,0,130,130)
draw_rect(130,50,80,80)
draw_rect(160,0,50,50)
draw_rect(130,0,30,30)
draw_rect(130,30,20,20)
draw_rect(150,40,10,10)
set_pen("medium","solid")
set_color(255,153,50)
z=130 ; draw_arc(0,0-z,2*z,2*z,90,90)
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)
```



3. Iteratieve grafische algoritmes

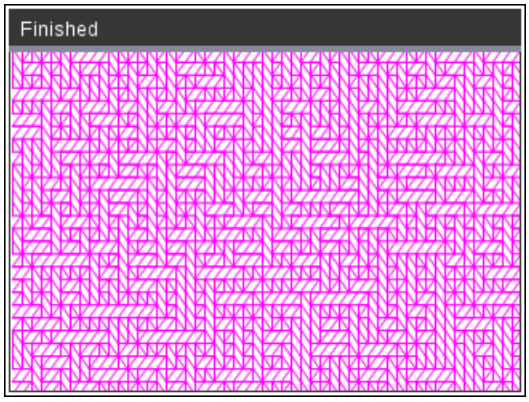
Voor de volgende voorbeelden gebruiken we de standaard vensterinstellingen.

3.1. At random creatief met tekst

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

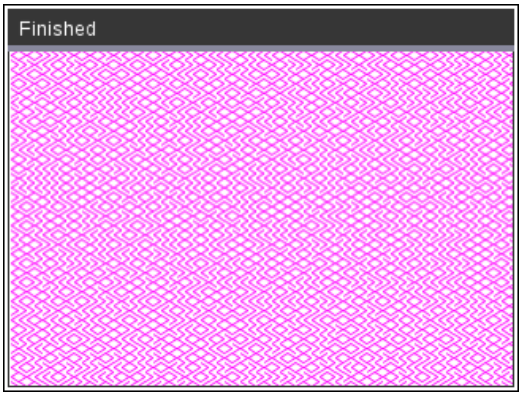
"N" of "Z"

```
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()
```

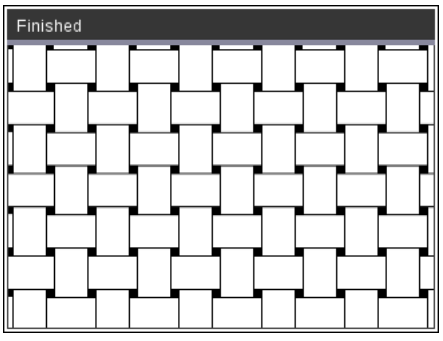


3.2. Modulo-matje weven

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:

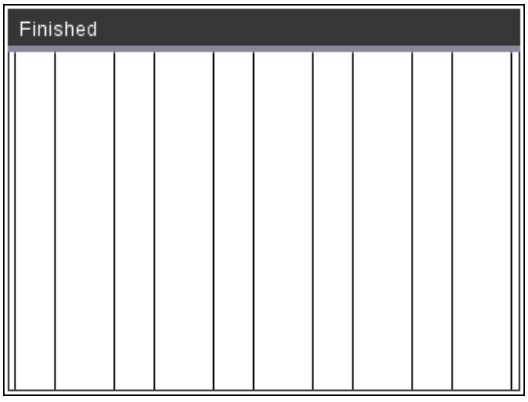
```
for i in range(0,318,d):
    x=i+s
    y=0
    draw_line(x,y,x,y+d)
paint_buffer()
```

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

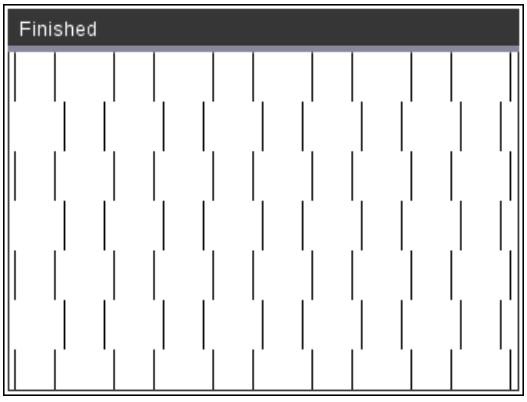
```
for i in range(0,318,d):
    if i%2 == 0:
        x=i+s
    else:
        x=i-s
    y=0
    draw_line(x,y,x,y+d)
paint_buffer()
```

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()
```

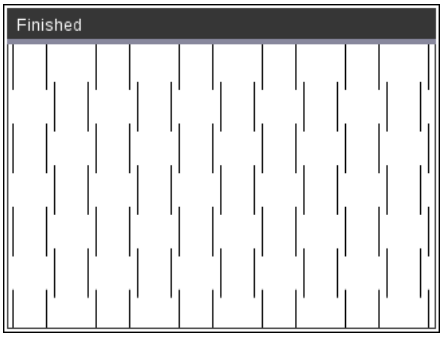



```
for j in range(0,212,d):
    for i in range(0,318,d):
        if (i+j)%2 == 0:
            x=i+s
        else:
            x=i-s
        y=j
        draw_line(x,y,x,y+d)
paint_buffer()
```



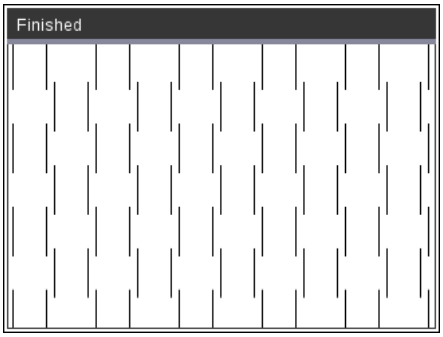
Als laatste stap verlengen we de lengte van de lijnstukkenmet 2*s, een afstand s omhoog en omlaag:

```
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()
```

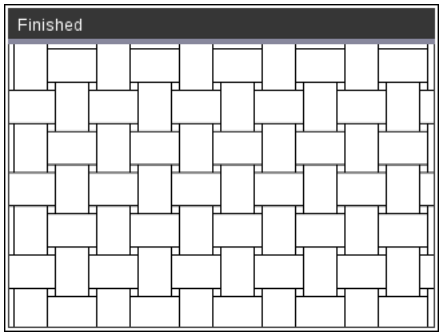


Voor de horizontale lijnstukken wisselen we de rollen van x en y om:

```
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()
```

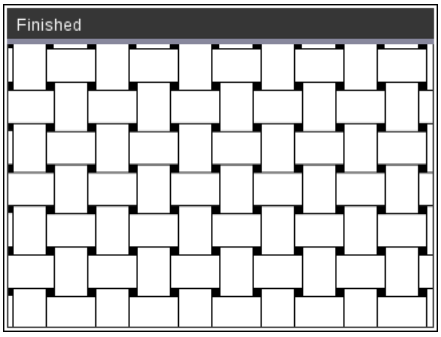


Beide blokken code voor de verticale en horizontale lijnstukken gecombineerd, geeft:



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()
```



3.3. Verborgen cirkels

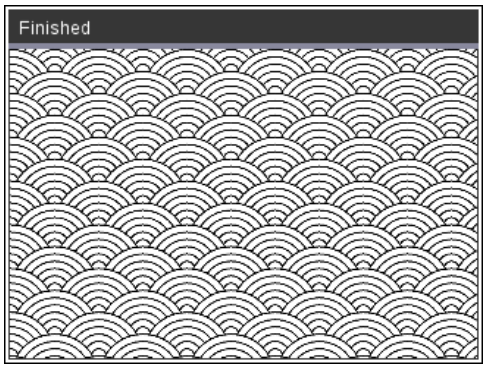
In het volgende voorbeeld wordt de kern van de herhalingen bepaald door zes cirkel met hetzelfde middelpunt en achtereenvolgend de stralen r = 30, r = 25, r = 20, r = 15, r = 10 en r = 5.

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

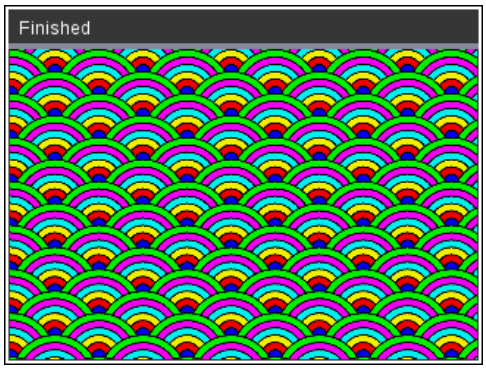
Voor de y-coordinaten verdelen we de hoogte van het scherm in 15.

```
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()
```



3.4. Optische misleidingen

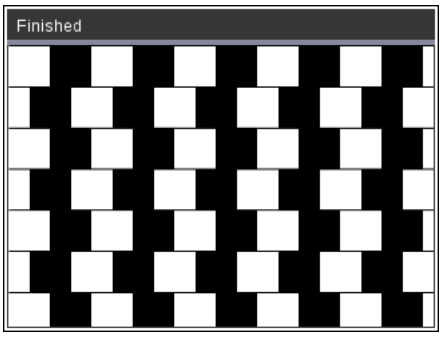
Voorbeeld 1
Parallelle vierkanten?

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

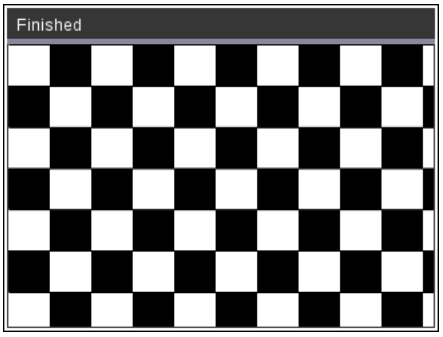
```
from ti_system import *.
from ti_draw import *
factor=float(input("Factor -1 ...  1: "))
use_buffer()
def vierkant(x,y,zijde):
    px=[x,x+z,x+z,x,x]
    py=[y,y,y+z,y+z,y]
    fill_poly(px,py)
z=31
set_color(255,255,255)
fill_rect(0,0,318,212)
set_color(0,0,0)
for j in range(8):
    draw_line(0,j*z,318,j*z)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

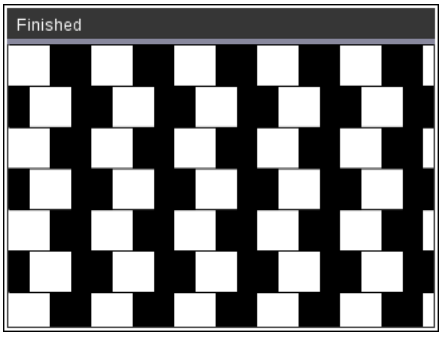
Factor = -0.5



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)

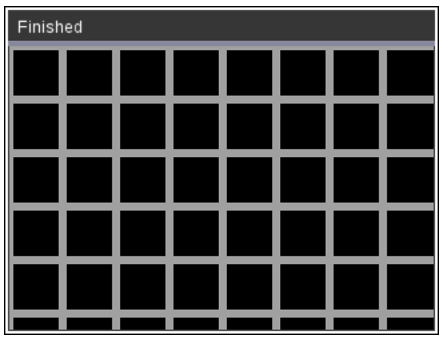
```
set_color(160,160,160)
for j in range(0,212,zijde):
♦♦fill_rect(0,j-b,318,2*b)
```

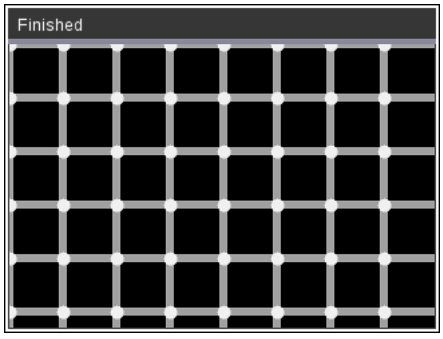
Verticale lijnen (grijs)

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

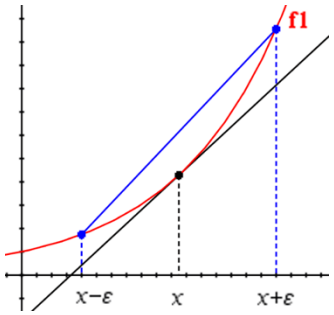
Stippen (wit)

```
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()
```





1. Afgeleide functie

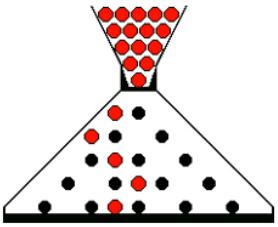


a. Plot de functie $f(x) = \sin(x)$ voor $x \in [-4\pi, 4\pi]$.

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:

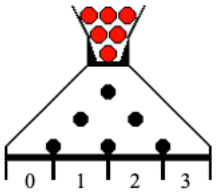
$$f'(x) = \frac{f(x+\varepsilon)-f(x-\varepsilon)}{2\varepsilon} \text{ met } \varepsilon = 0.003.$$

2. Kans-simulatie



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

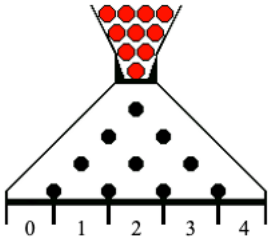
Wanneer een bal een pin raakt is de kans dat de bal links rolt gelijk aan de kans dat de bal rechts rolt.



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

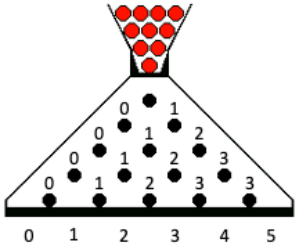
Het bord heeft een binomiale kansverdeling X ~ B(3,½): P(X=0) = $\frac{1}{8}$ P(X=1) = $\frac{3}{8}$ P(X=2) = $\frac{3}{8}$ P(x=3) = $\frac{1}{8}$



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.

3. Lineaire regressie

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

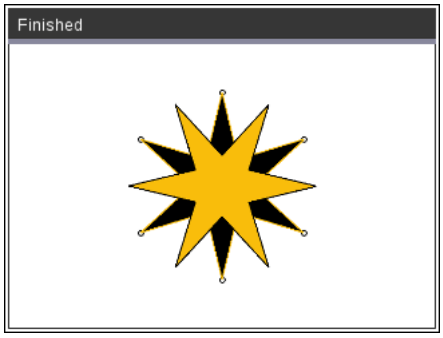
lengte = [53,52,53,56,51,51,54,53,50,53,50,52,53,52,52]
gewicht = [4.22,3.08,3.76,4.75,3.54,3.54,4.13,3.76,2.8,3.49,2.86,3.13,3.18,3.31,3.9]

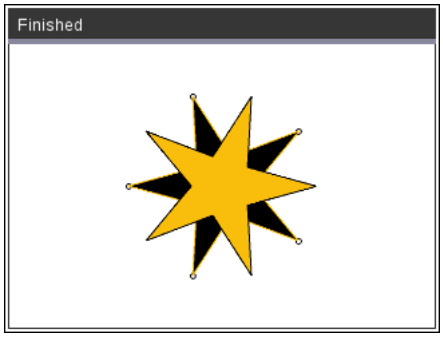
4. Fotobewerking

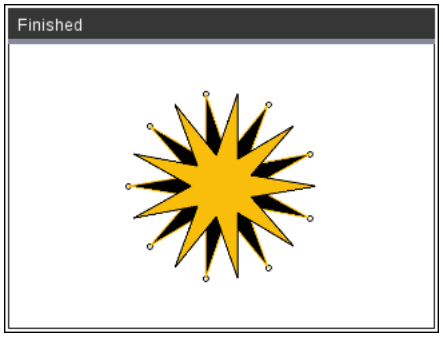
Upload je favoriete foto in een Notes-applicatie van de TI-Nsprire CX-software en pas een aantal manipulaties & filters uit de TI Python BootCamp toe.

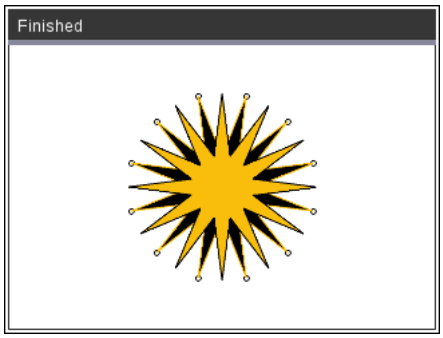
5. Ster

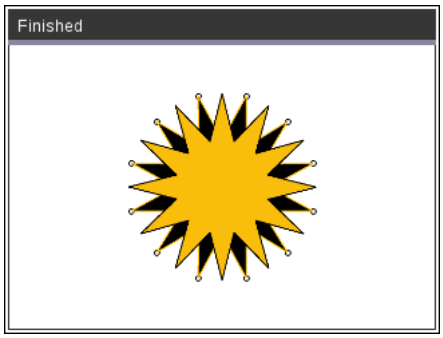
Teken/codeer de onderstaande ster en gebruik variabelen om de onderstaande varianten te genereren.

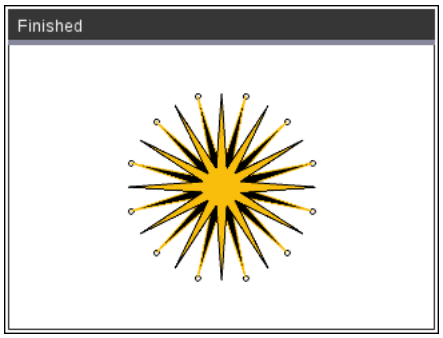


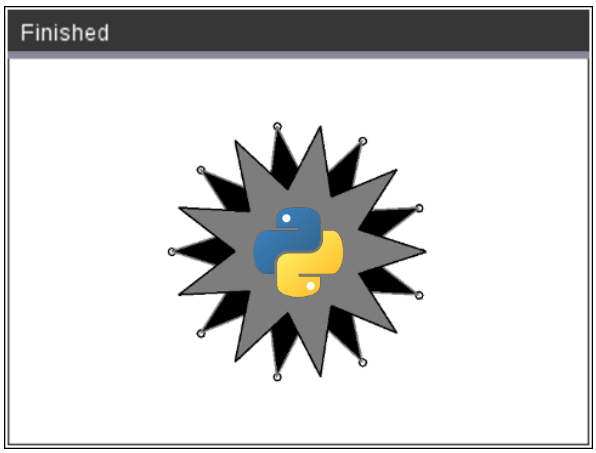










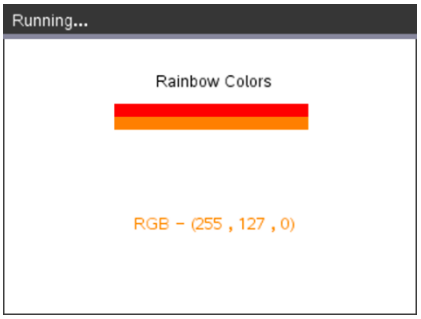


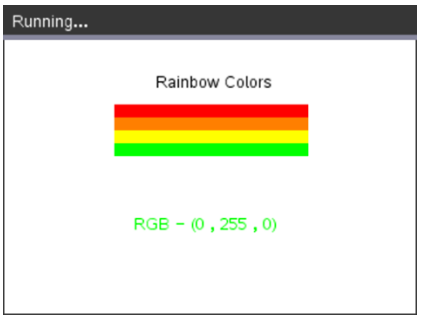
1. Dashboard

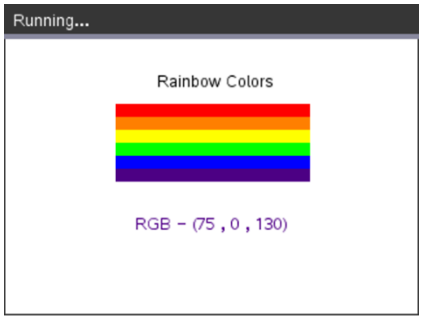
Gebruikmakend van de grafische modules kunnen we samen met het sturen en lezen van de TI-Innovator hub een grafisch dashboard coderen/genereren.

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)
```

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

2.1. Keyboard-input

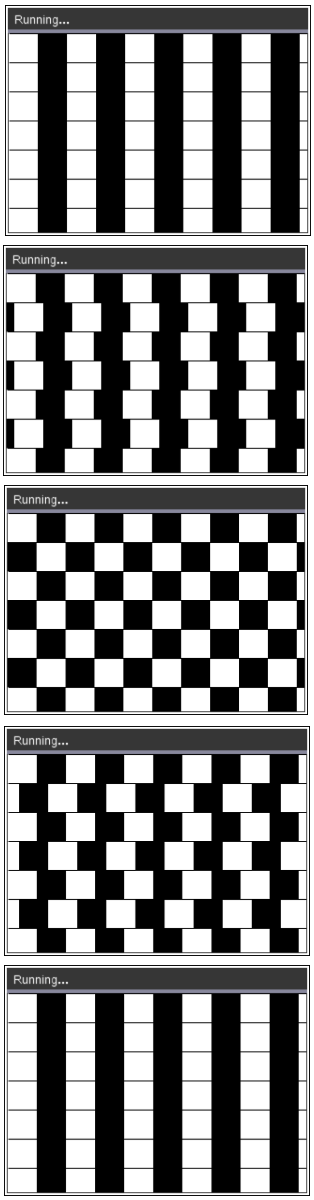
Voor het coderen van de onderstaande *Optische misleiding* veranderen we de variabele factor met behulp van het get_key()-statement (van de TI System-module) in een while-lus. We verminderen of vermeerderen factor met 0.05 door respectievelijk het intikken van ← en →.

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)
```



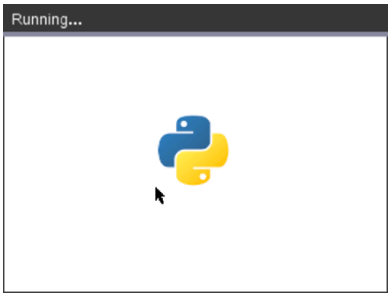
Merk op dat indien we het statement get_key(1) uitvoeren dat de code stopt met runnen tot dat een toets wordt ingedrukt.

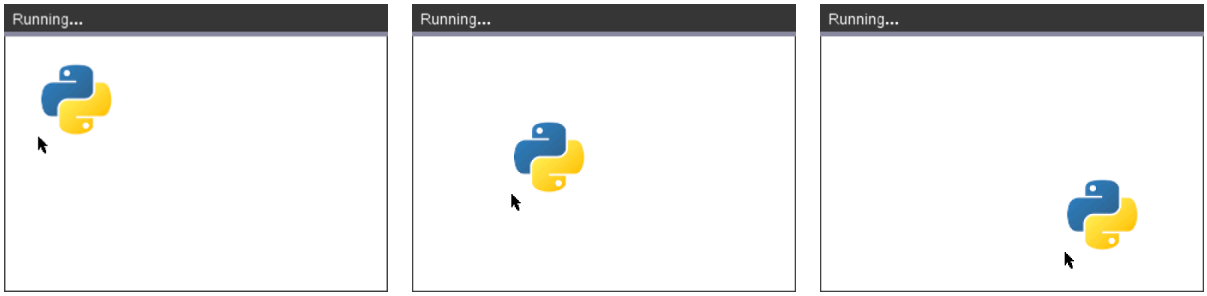
1.1. Muis-positie

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().

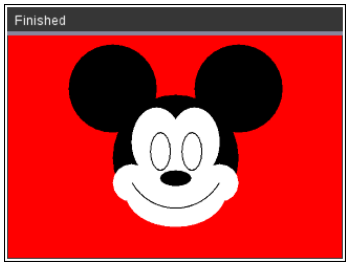
```
from ti_image import *
from ti_system import *
img=load_image("py")
use_buffer()
while get_key()!="scherm":
    clear()
    x,y=get_mouse()
    img.show_image(x,y-img.h)
    paint_buffer()
```





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)
```

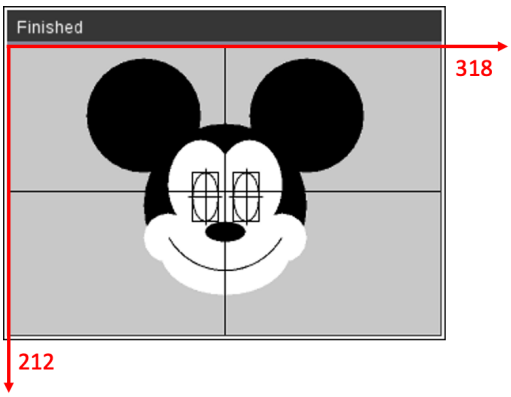



```
def mickey():
    set_color(255,0,0)
    fill_rect(-dim[0]/2,-dim[1]/2,318,212)
    set_color(0,0,0)
    fill_circle(0,-10,60)
    fill_circle(-60,56,42)
    fill_circle(60,56,42)
    set_color(255,255,255)
    fill_arc(-48,-76,96,66,0,360)
    fill_circle(42,-34,18)
    fill_circle(-42,-34,18)
    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)
```

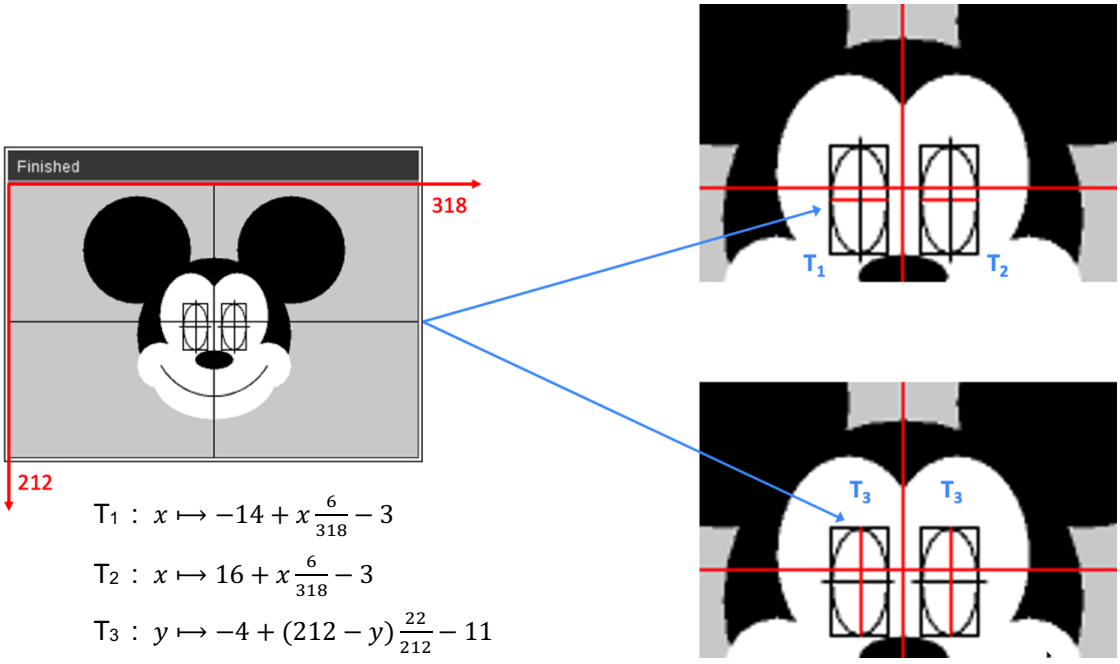
Met get_mouse() gaan we zorgen dat de pupillen de bewegingen van de cursor volgen.

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:



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

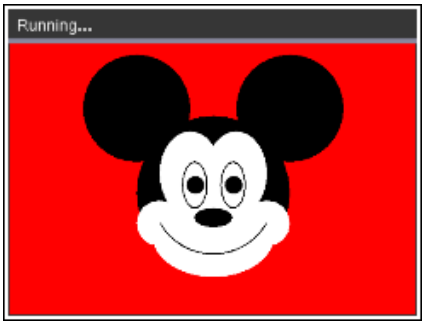
$$T_2 : x \mapsto 16 + x\frac{6}{318} - 3$$

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

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

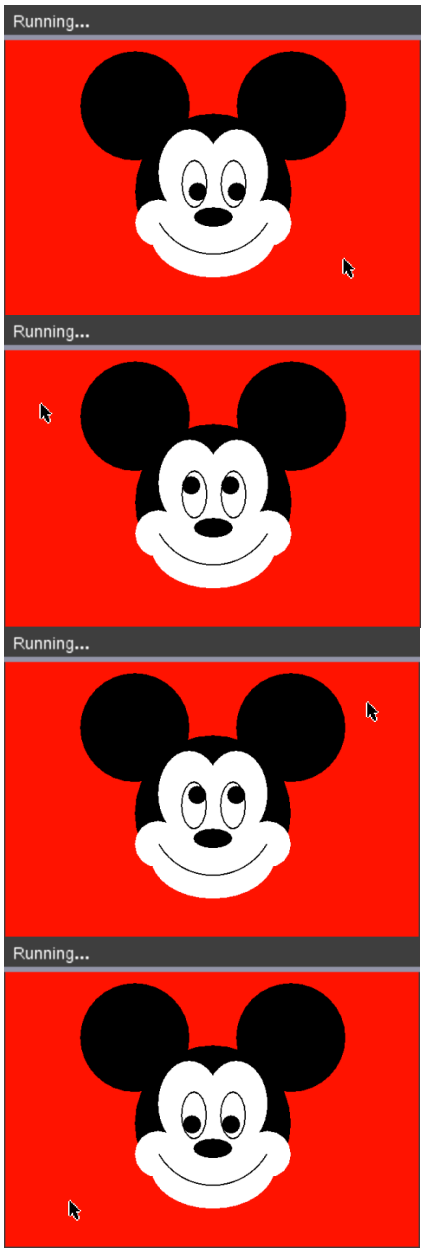
En T_3 beeldt het segment [0,212] af op [-15,7], samen met het veranderen van de positieve oriëntatie.

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()
```



2. Een digitale analoge klok

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

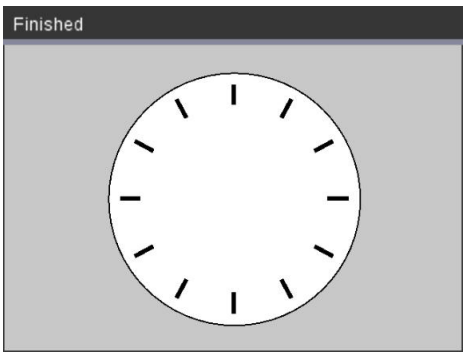
Het statement localtime() van de time-module creëert een tuple met de volgende data:
(jaar , maand , dag , uur, minuut , seconde , weekdag , dagnummer, Daylight Savings indicator).

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

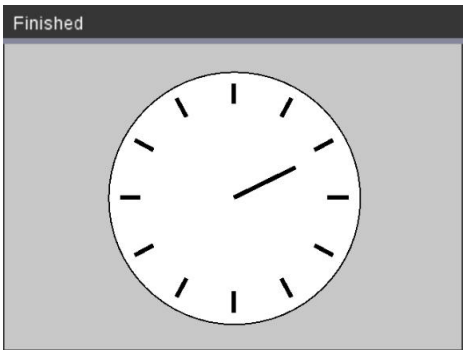
Samen met de onderstaande grafische definities bouwen we digitaal een analoge klok.

```
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()

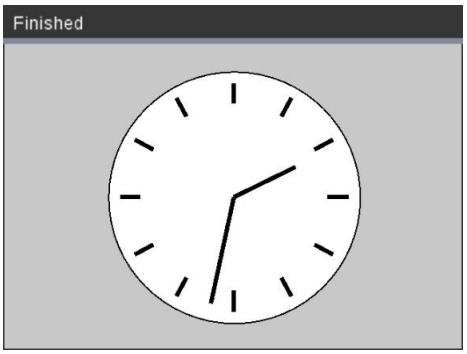


hour_hand(14,11)



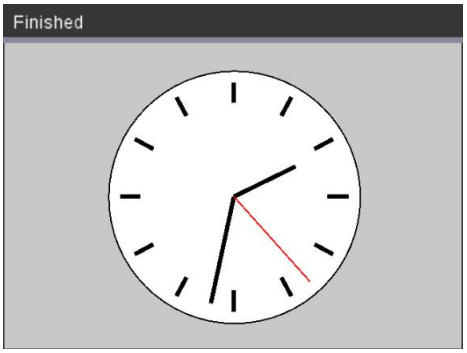
```
def minute_hand(m):
    a=pi/2-m*pi/30
    x=0.95*r*cos(a)
    y=0.95*r*sin(a)
    draw_line(0,0,x,y)
```

minute_hand(32)

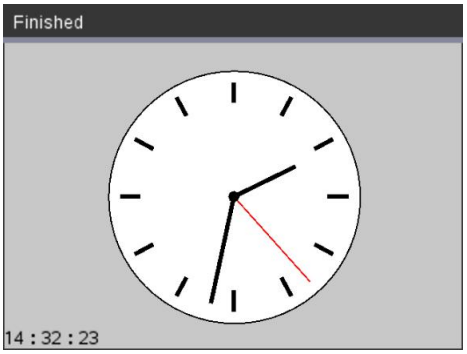



```
def second_hand(s):
    set_color(255,0,0)
    a=pi/2-s*pi/30
    x=r*cos(a)
    y=r*sin(a)
    draw_line(0,0,x,y)
```

second_hand(23)

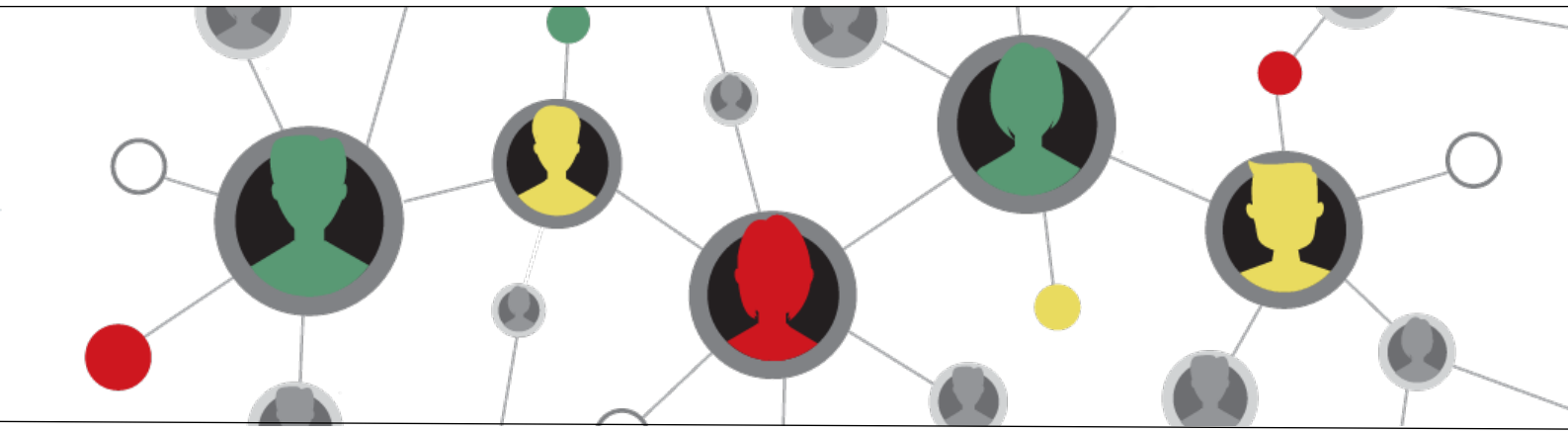



```
while get_key() != "esc":
    clear()
    draw_clock()
    t=localtime()
    hr,min,sec=t[3],t[4],t[5]
    draw_text(-3,-2,"{} : {} : {}".format(hr,min,sec))
    set_color(0,0,255)
    hour_hand(hr,min)
    minute_hand(min)
    set_color(255,0,0)
    set_pen(0,0)
    second_hand(sec)
    set_color(0,0,0)
    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:1

```
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
```



T³ NEDERLAND T³ VLAANDEREN

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

Texas Instruments