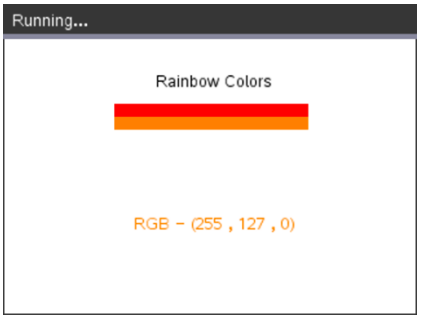
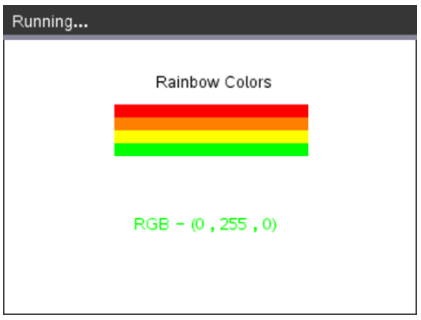
1. Dashboard

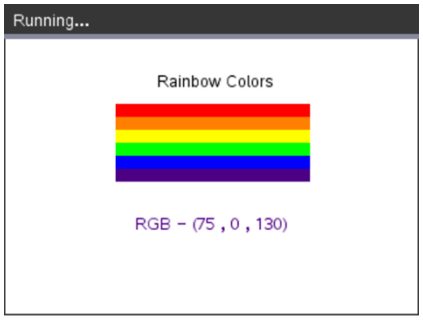
Gebruikmakend van de grafische modules kunnen we samen met het sturen en lezen van de TI-Innovator hub een grafisch dashboard coderen/genereren.

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

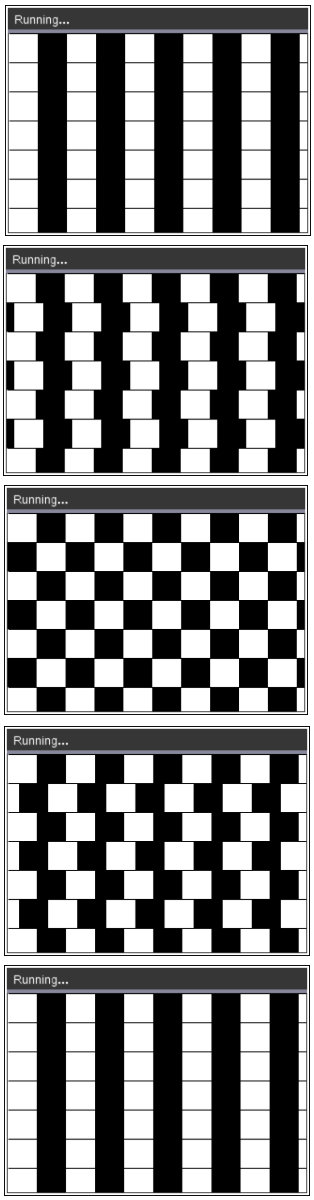
Voor het coderen van de onderstaande *Optische misleiding* veranderen we de variabele factor met behulp van het get_key()-statement (van de TI System-module) in een while-lus. We verminderen of vermeerderen factor met 0.05 door respectievelijk het intikken van ← en →.

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)
```



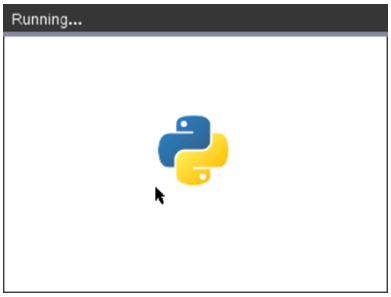
Merk op dat indien we het statement get_key(1) uitvoeren dat de code stopt met runnen tot dat een toets wordt ingedrukt.

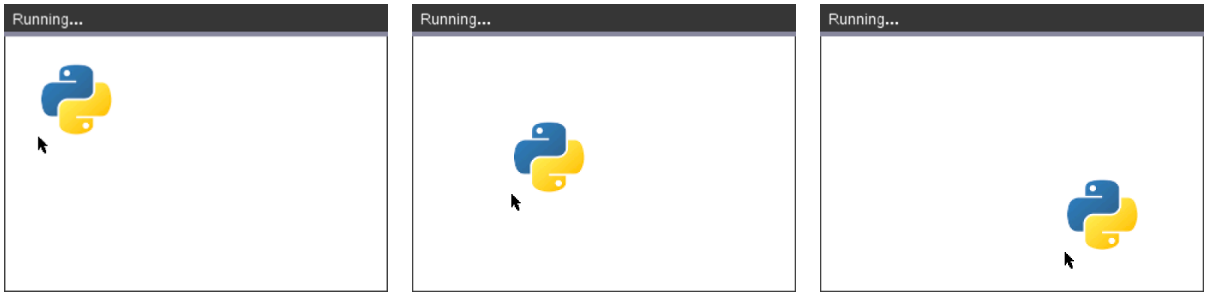
1.1. Muis-positie

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().

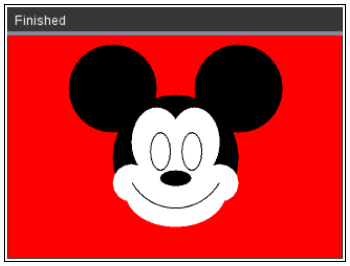
```
from ti_image import *
from ti_system import *
img=load_image("py")
use_buffer()
while get_key()!="scherm":
    clear()
    x,y=get_mouse()
    img.show_image(x,y-img.h)
    paint_buffer()
```





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)
```

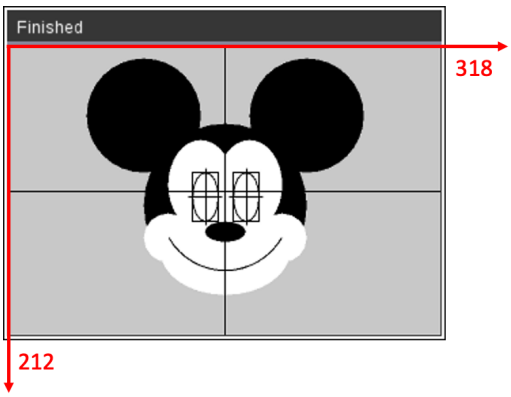



```
def mickey():
    set_color(255,0,0)
    fill_rect(-dim[0]/2,-dim[1]/2,318,212)
    set_color(0,0,0)
    fill_circle(0,-10,60)
    fill_circle(-60,56,42)
    fill_circle(60,56,42)
    set_color(255,255,255)
    fill_arc(-48,-76,96,66,0,360)
    fill_circle(42,-34,18)
    fill_circle(-42,-34,18)
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

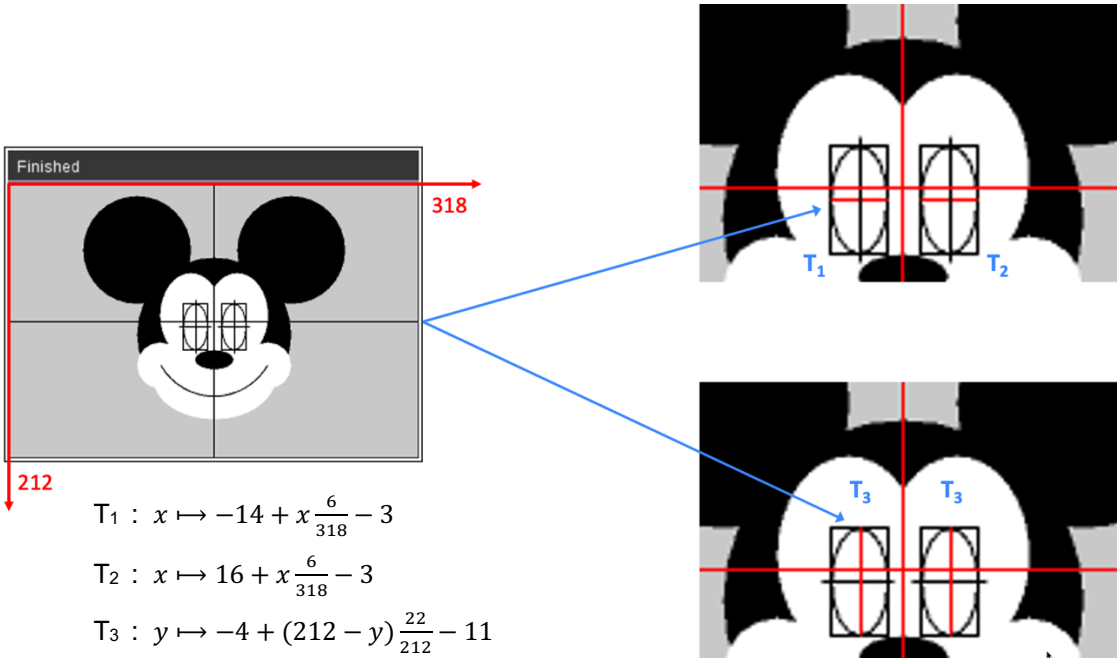
Met get_mouse() gaan we zorgen dat de pupillen de bewegingen van de cursor volgen.

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:



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

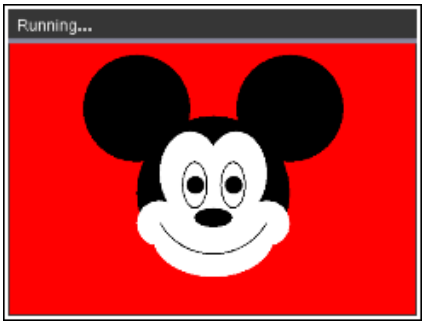
$$T_2 : x \mapsto 16 + x\frac{6}{318} - 3$$

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

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

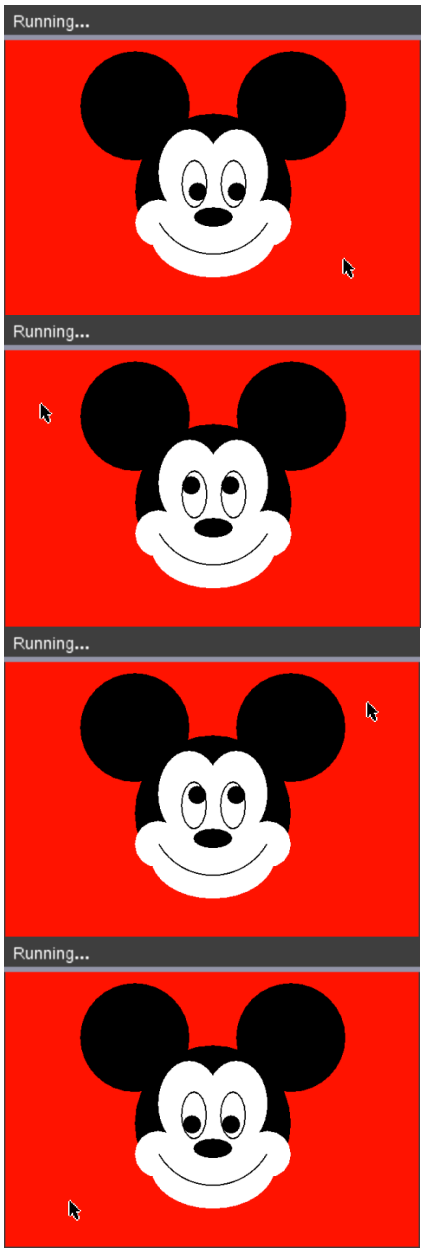
En T_3 beeldt het segment [0,212] af op [-15,7], samen met het veranderen van de positieve oriëntatie.

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.

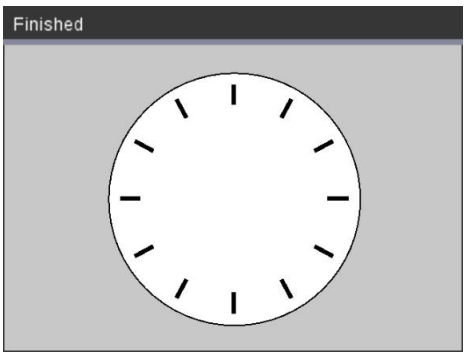
Het statement localtime() van de time-module creëert een tuple met de volgende data:
(jaar , maand , dag , uur, minuut , seconde , weekdag , dagnummer, Daylight Savings indicator).

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

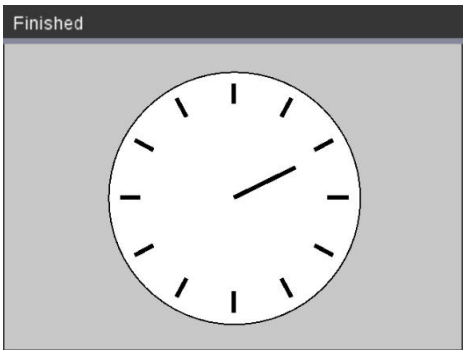
Samen met de onderstaande grafische definities bouwen we digitaal een analoge klok.

```
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()

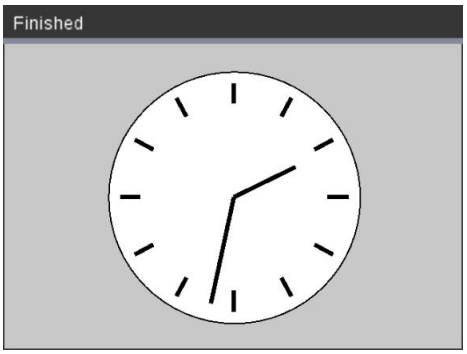


hour_hand(14,11)



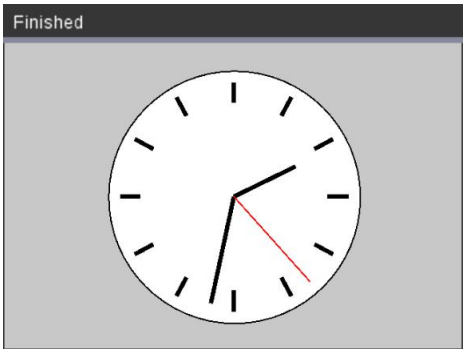
```
def minute_hand(m):
    a=pi/2-m*pi/30
    x=0.95*r*cos(a)
    y=0.95*r*sin(a)
    draw_line(0,0,x,y)
```

minute_hand(32)

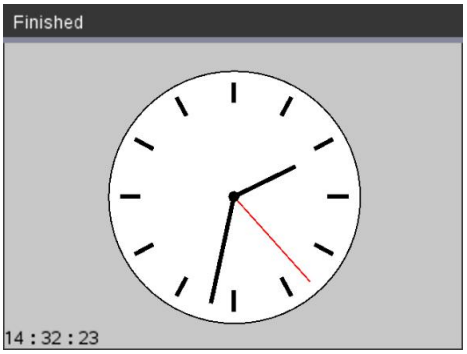



```
def second_hand(s):
    set_color(255,0,0)
    a=pi/2-s*pi/30
    x=r*cos(a)
    y=r*sin(a)
    draw_line(0,0,x,y)
```

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
```