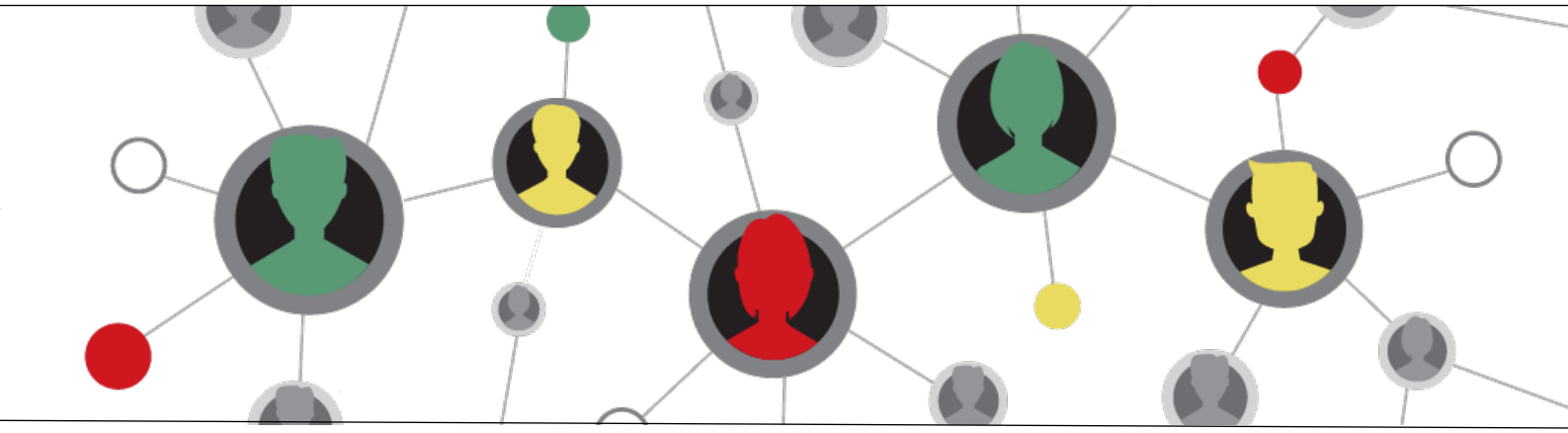
Computational Thinking

Programmeerboekje

TI-84 Plus CE-T PYTHON EDITION



Didier Deses

T³ Vlaanderen

T³ NEDERLAND

Teachers Teaching with Technology™

T³ VLAANDEREN

Inhoudsopgave

1 Introductie tot Python op de TI-84+ 3

1.1 De Python App ... 3
1.2 De basis van programmeren ... 5
1.2.1 Voorwaardelijke structuur ... 5
1.2.2 Lussen ... 7
1.3 De grafische module ... 8

2 Voorbeeldprogramma's 10

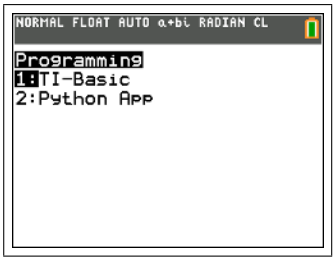
2.1 Programmeertrucjes ... 10
2.2 Enkele korte programma's (1 scherm lang) ... 12
2.2.1 Een eenvoudig spel: 24 ... 12
2.2.2 Schaar-Steen-Papier ... 13
2.2.3 Galgje ... 14
2.2.4 Doolhof ... 15
2.2.5 Cursor door een doolhof ... 16
2.2.6 Random walks ... 17
2.2.7 Beurskoersen ... 18
2.2.8 Galton bord ... 19
2.2.9 Binaire getallen ... 20
2.2.10 Getallen in een andere basis. ... 21
2.2.11 Caesar-code ... 22
2.2.12 Sterren ... 23
2.2.13 String art ... 24
2.2.14 Horner ... 25
2.2.15 Veeltermproducten ... 26
2.2.16 Grafieken van functies in 3d ... 27
2.2.17 Thomae's functie ... 28
2.2.18 De driehoek van Pascal en de zeef van Sierpinski ... 29
2.2.19 IFS ... 30
2.2.20 Julia-fractalen: Backtracking methode ... 31
2.2.21 Model van Verhulst en chaos ... 32
2.2.22 Turtle graphics: een eigen programmeertaalmaken! ... 33
2.2.23 De Koch kromme ... 35

Hoofdstuk 1

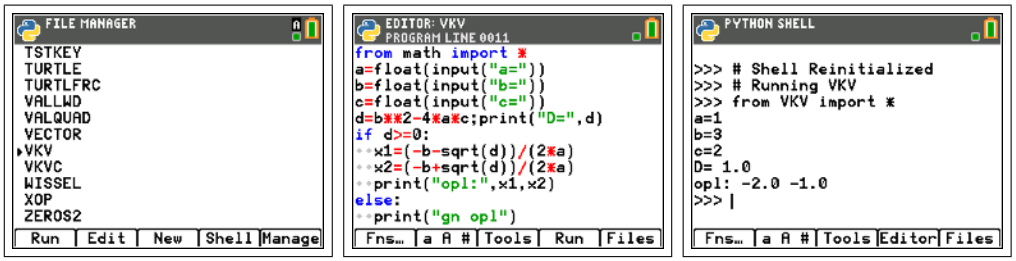
Introductie tot Python op de TI-84+

1.1 De Python App

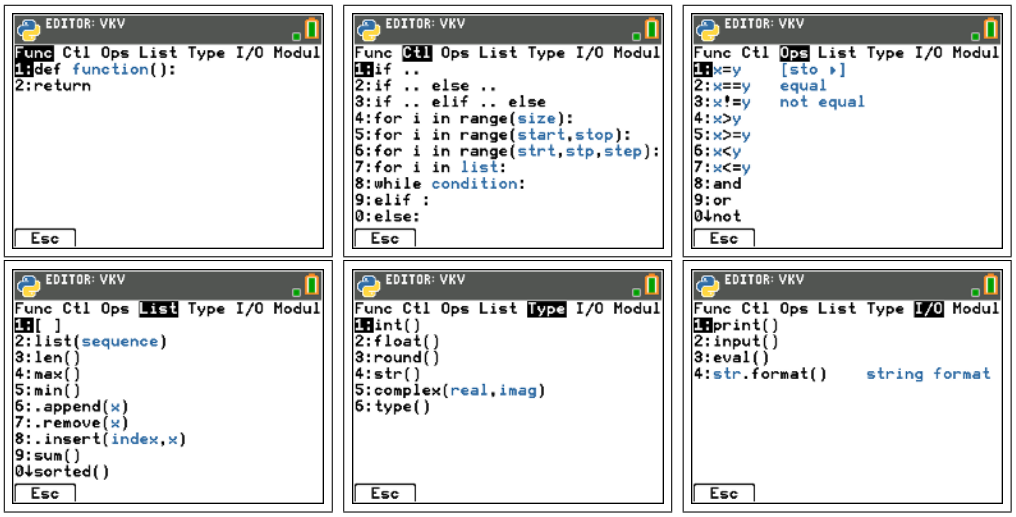
Rekenmachine 1 (Referentie Python op de **TI-84+**). Op de moderne **TI-84+** heeft men de keuze tussen twee programmeertalen, druk maar eens op `prgm`.



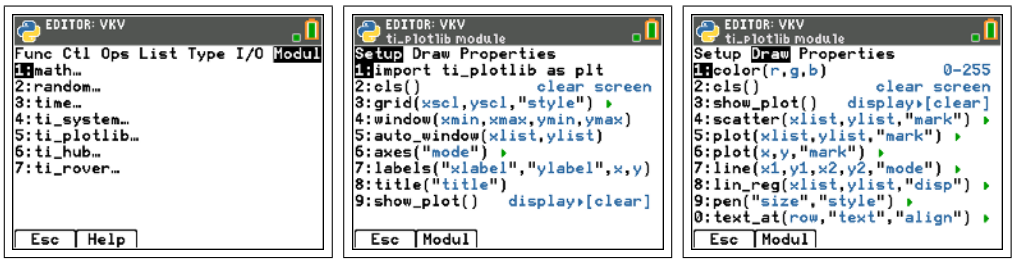
Na het starten van de `Python App` krijg je de `file manager` te zien. Deze biedt een overzicht van de programma's die reeds op het toestel staan en via de functietoetsen de mogelijkheid om een nieuw programma te maken of een bestaand programma aan te passen. Het schrijven van een programma gebeurt in de `editor` van de app. Wanneer we het programma laten draaien (`[Run]`) wordt een `Python shell` gestart waarin het programma uitgevoerd wordt.



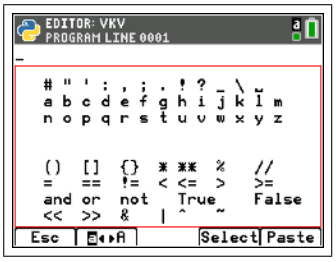
We geven een kort overzicht van de menu's die relevant zijn voor het programmeren. In de `editor` kan je via `[Fns...]` volgende commando's terugvinden, mooi gerangschikt per functionaliteit.



Andere Python commando's vind je via 2nd [catalog]. Delen van Python worden als afzonderlijke module opgeladen. Zo zitten alle wiskundige functies in de module `math`. De module wordt geladen door `from math import *`, het commando `sqrt` kan vanaf dan gebruikt worden (zie bovenstaand voorbeeld `vkv`). Naast tal van andere modules is ook de module `ti_plotlib` beschikbaar. Deze dient om de grafische mogelijkheden van de **TI-84+** te gebruiken in Python. De module wordt standaard geladen via `import ti_plotib as plt` zodat alle grafische commando's beginnen met `plt`.



Zoals te zien aan de commando's in het menu-systeem is de Python versie op de **TI-84+** redelijk uitgebreid qua mogelijkheden en dan staan we nog niet stil bij modules zoals `ti_rover` die in staat zijn om een autootje te programmeren en te besturen! Voor de volledigheid geven we nog mee dat in de `editor` het menu `[a A ]` extra symbolen geeft zoals de puntkomma.



1.2 De basis van programmeren

Programma 1 (Hello World)**.** Traditioneel is het eerste programma dat je schrijft in een programmeertaal het programmma *Hello World*, dat gewoon deze boodschap op het scherm afdrukt. In Python is dit uiterst eenvoudig en bestaat het programma uit één regel Probeer maar!

Opmerking 2. Elke programmeertaal heeft naast variabelen en bewerkingen twee controle-structuren:

- voorwaardelijke structuren
- lussen

1.2.1 Voorwaardelijke structuur

Opmerking 3. De **voorwaardelijke structuur** geeft de mogelijkheid om een deel van de code slechts uit te voeren wanneer er een bepaalde voorwaarde

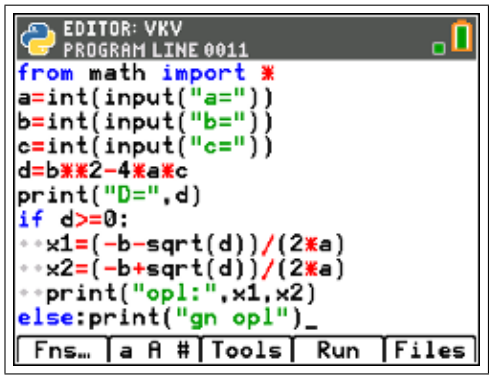
geldt. In Python gebeurt dit met de `if`-structuur . Deze heeft verschillende vormen, waaronder:

```
if voorwaarde:
    commando's


If voorwaarde:
    commando's
else:
    commando's
```

Merk op dat de uit te voeren commando's telkens gegroepeerd worden door in te springen (**indentatie**).

Programma 2 (Vierkantsvergelijkingen)**.** We geven een kort voorbeeld van een Python-programma om vierkantsvergelijkingen op te lossen in $\mathbb{R}$.



We bespreken dit kort:

- In Python zitten de wiskundige functies zoals `sin()`, `cos()`, `tan()`, `sqrt()`, `...` in de aparte module `math`. Met de regel `from math import *` worden al deze functies beschikbaar.
- In de volgende drie regels worden de coëfficiënten opgevraagd. Omdat `input()` een string teruggeeft, moet deze eerst omgezet worden naar een geheel getal d.m.v. `int()`.
- Hierna wordt de discriminant berekend. Merk op dat het kwadraat b^2 wordt ingegeven als `b**2`.
- Met een `if`-structuur wordt een onderscheid gemaakt tussen de gevallen $D \geq 0$ en $D < 0$.

- In het eerste geval worden de oplossingen x_1 en x_2 berekend en afgeprint.
- In het andere geval wordt afgedrukt dat er geen oplossingen zijn.

1.2.2 Lussen

Opmerking 4. De tweede structuur die elke programmeertaal bezit, zijn **lussen**. De eerste is de `for`-lus :

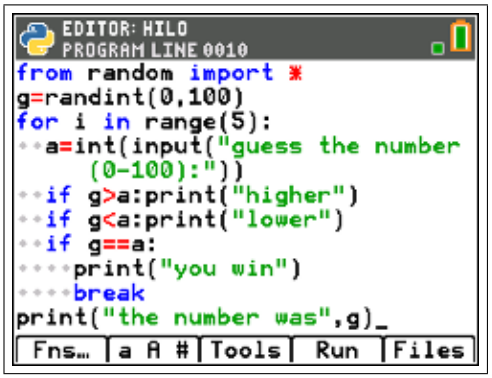
```
for i in range(n):
    commando's


for i in range(a,b):
    commando's
```

De commando's zullen voor elke waarde van de variabele i vanaf 0 tot $n-1$ uitgevoerd worden. In de tweede versie krijgt i waarden van a tot $b-1$. Hierbij is het noodzakelijk dat alle parameters gehele getallen zijn.

Wanneer een reeks commando's herhaald worden in een lus spreekt men van **iteratie**.

Programma 3 (Hoger-Lager)**.** We geven een uitgewerkt voorbeeld van het spelletje 'hoger-lager'.



We bespreken dit kort.

- De commando's om willekeurige getallen te genereren staan in de module `random`. We importeren deze functies.
- Na het afdrukken van een titel kiest de computer een willekeurig geheel getal tussen 1 en 100 d.m.v. `randint()`.

- Nu begint een `for`-lus die de speler vijf beurten geeft.
- Na de input van de speler wordt twee keer met een `if`-structuur de juiste tip gegeven.
- Indien de speler gewonnen heeft, wordt met `break` de `for`-lus onmiddellijk stopgezet en eindigt het programma.

1.3 De grafische module

Rekenmachine 5 (Grafische module)**.** Op de **TI-84+** wordt de module `ti_plotlib` meegeleverd. Hiermee kan je de grafische capaciteiten aanspreken.

- `ti_plotlib`:
  - Het gemakkelijkst is om deze te laden via `import tiplotlib as plt`. Dan vind je via menu de verschillende commando's.
  - `plt.window(xmin,xmax,ymin,ymax)`: stelt het assenkruis in. Met `plt.xmin`, `plt.xmax`, ... kan je de grenzen later opvragen.
  - `plt.cls()`: veegt het scherm leeg.
  - `plt.axes("on")`: tekent de assen en zet er de grenzen bij. Met de parameter `"axes"` of gewoon `plt.axes()` worden enkel de assen getekend.
  - `plt.grid(xscl,yscl,"solid")`: tekent een grid over het grafisch scherm.
  - `plt.plot(x,y,".")`: tekent een dun punt op het scherm, met `"o"` wordt het een dik punt.
  - `plt.line(x1,y1,x2,y2,"")`: trekt een lijn van (x_1, y_1) naar (x_2, y_2). Met `"arrow"` kan je ook een pijl tekenen.
  - `plt.color(r,g,b)`: gebruikt r, g, b-waarden om de kleur in te stellen.

Programma 4. Een eenvoudig programma om de grafiek van een functie te tekenen kan als volgt gemaakt worden.

- Eerst worden de modules `ti_plotlib` en `math` geladen.
- De assen van het grafisch venster worden ingesteld en getekend nadat het scherm gewist is.
- Het aantal punten dat we tekenen is n. De x-as wordt dus verdeelt in n stukken. Een stapje op de x-as komt dus overeen met een lengte dx.
- Met een `for`-lus worden de n punten (x, y) van de grafiek berekend.
- Met `plt.plot(x,y,".")` tekenen we telkens het berekende punt.

Hoofdstuk 2

Voorbeeldprogramma's

2.1 Programmeertrucjes

Opmerking 6. We geven enkele voorbeeldprogramma's. Merk op dat we hier soms zondigen tegen de *Zen of Python* in die zin dat alle programma's zo geschreven zijn dat ze op één enkel schermpje passen. Hiervoor is het soms nodig om meerdere opdrachten op één lijn te zetten, gescheiden door `;`. Het zo kort mogelijk schrijven van een bepaald stuk code noemt men *code golfing*. Hiervoor worden heel wat programmeertrucjes gebruikt om code in te korten of efficiënter te maken. We bespreken er hier enkele van.

Opmerking 7 (Definiëren van meerdere variabelen)**.** Wanneer meerdere variabelen een waarde moeten krijgen gebruikt men volgende code.

```
a=1
b=-1
c=-1
```

Dit kan in één enkele regel als volgt:

```
a,b,c=1,-1,-1
```

Opmerking 8 (Lijsten initialiseren)**.** We zullen vaak `[0]*5` gebruiken om de lijst `[0,0,0,0,0]` aan te maken.

Opmerking 9 (Importeren van modules)**.** Je kan modules importeren op twee manieren

```
import ... as ...
```

```
from ... import *
```

Je gebruikt de commando's uit de module dan als volgt:

```
import ti_plotlib as plt ----> plt.cls(), plt.plot(x,y)

from ti_plotlib import * ----> cls(), plot(x,y)
```

De eerste versie zorgt voor duidelijkheid maar de code neemt veel plaats in. Met de tweede vorm kan je de commando's onmiddellijk gebruiken. Dit maakt de code een stuk korter. Let wel, de menu's van de **TI-84+** gebruiken standaard de eerste notatie.

Opmerking 10 (Verkort kleurenpalet)**.** Python gebruikt r, g, b-waarden om kleuren te encoderen. Elk van deze waarden zitten tussen 0 en 255, er zijn dus $256^3 \approx 16$ miljoen kleuren beschikbaar. Vaak is het echter zo dat we slechts enkele contrasterende kleuren willen gebruiken. Je kan dan een palet samenstellen.

```
zrgb=[(0,0,0),(255,0,0),(0,255,0),(0,0,255)]
```

De lijst van tupels `zrgb` bevat de kleuren zwart, rood, groen en blauw. Met `plt.color(zrgb[1])` kan je dus rood selecteren. Een volledig kleurenpalet ingeven op de **TI-84+** is nogal onhandig. We kunnen dit eenvoudiger. Uit het tupel

```
co=(0,0,0,255,0,0)
```

worden telkens drie opeenvolgende elementen geselecteerd d.m.v. `co[k%4:k%4+3]`. Dit geeft de volgende kleuren:

$$k = 0 : (\underbrace{0, 0, 0}_{\text{zwart}}, 255, 0, 0)$$

$$k = 1 : (0, \underbrace{0, 0, 255}_{\text{blauw}}, 0, 0)$$

$$k = 2 : (0, 0, \underbrace{0, 255, 0}_{\text{groen}}, 0)$$

$$k = 3 : (0, 0, 0, \underbrace{255, 0, 0}_{\text{rood}})$$

Op deze wijze bekomen we dezelfde vier kleuren als voorheen. Met onderstaande tupels kan je op analoge wijze van drie tot acht kleuren bekomen.

```
col3=[0,0,255,0,0]
col4=[0,0,0,255,0,0] #dit is bovenstaande co
col6=[0,0,255,0,255,255,0,0]
col7=[0,0,0,255,0,255,255,0,0]
col8=[0,0,0,255,0,255,255,255,0,0]
```

2.2 Enkele korte programma's (1 scherm lang)

2.2.1 Een eenvoudig spel: 24

Neem uit een kaartspel de cijferkaarten (1,2,...,9). Meng de kaarten en draai de eerste vier kaarten om, de eerste speler die met de bewerkingen $+, -, \cdot, /$ de waarde 24 kan maken wint de ronde.

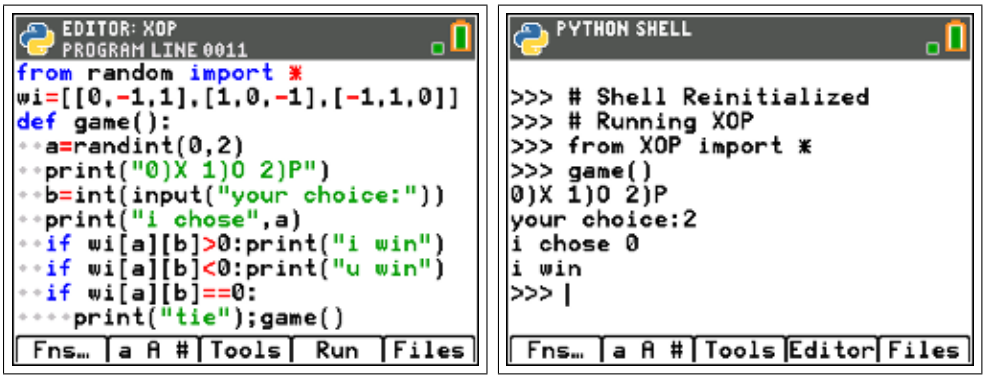
Zoals je ziet is dit een erg eenvoudig wiskundig spel dat soms in het basis- of middelbaar onderwijs gebruikt wordt om hoofdrekenen te bevorderen. Er bestaan varianten die ook de hogere kaarten toelaten (10, boer=11, ...). Bovendien kan het ook als solitaire spel gespeeld worden.

De eenvoud van het spel vertaalt zich ook in een eenvoudig Pyhton-programma:

Wat de code zo eenvoudig maakt is `eval(input())`. Dit staat toe dat een gebruiker een formule ingeeft, door Python geëvealueerd wordt.

2.2.2 Schaar-Steen-Papier

Onderstaand programma laat je het spel Schaar-Steen-Papier op de **TI-84+** spelen.



- Eerst wordt de module `random` geladen.
- We maken een matrix `wi` die de eindstand van alle spelletjes bevat. We stellen schaar-steen-papier voor door X-O-P en de overeenstemmende getalwaarden 0-1-2. Als de eerste speler bijvoorbeeld schaar heeft gekozen (waarde $a = 0$) kijken we in het nulde element van `wi`: `[0,-1,1]`, heeft de tweede speler papier gekozen (waarde $b = 2$) dan bekomen we het laatste element uit de rij: 1 en weten we dat de eerste speler heeft gewonnen. M.a.w. `wi[a][b]` geeft aan of speler a wint of verliest.
- Vervolgens definiëren we de procedure. Eerst maakt het programma de keuze a. Daarna worden de opties afgedrukt en kan de speler zijn keuze ingeven.
- Met drie `if`-structuren wordt de uitslag van het spel bepaald. In geval van gelijk spel wordt de procedure opnieuw aangeroepen.

In de tv-reeks *"The big bang theory"* komt de variant *"Rock, Paper, Scissors, Lizard, Spock "* voor. Kan je dit spel ook programmeren?

Scissors cuts Paper
Paper covers Rock
Rock crushes Lizard
Lizard poisons Spock
Spock smashes Scissors
Scissors decapitates Lizard

Lizard eats Paper
Paper disproves Spock
Spock vaporizes Rock
(and as it always has) Rock crushes Scissors

2.2.3 Galgje

We zullen het klassieke woordspel 'Galgje' in Python programmeren. Eén speler geeft een geheime woord in, de andere speler speelt het spel.

- We vragen eerst een geheim woord op. Dit woord wordt met `list()` omgezet naar een lijst van letters, bijvoorbeeld `['s','i','n','u','s']`.
- We drukken hierna twintig witregels af, zodat het ingevoerde woord van het scherm verdwijnt. Nu kan het toestel doorgegeven worden aan de andere speler.
- De lijst g bevat evenveel `"-"` als er letters zijn in het geheime woord. k zal het aantal foute gokken tellen en met `m="-O<-<"` zullen we een opgeknoopt mannetje tekenen (horizontaal).
- Met een `while`-lus gaan we de beurten af tot er meer dan de toegelaten fouten zijn gemaakt. Met `"".join(g),"\n",m[:k]` drukken we drie dingen af. Met de methode `"".join()` van de lege string, worden alle tekens uit g toegevoegd aan `""` en wordt g op deze wijze afgedrukt. Met `"\n"` beginnen we een nieuwe regel en met `m[:k]` worden de eerste k tekens van het opgeknoopt mannetje afgedrukt.
- Indien g en s gelijk zijn, is het geheime woord geraden en wint de speler. Met `break` springen we uit de lus en stopt het programma.
- We vragen nu aan de speler een nieuwe letter c. Indien de letter c in s zit, worden al deze letters uit het geheime woord overgezet naar g aan de hand van een `for`-lus . Indien c niet voorkomt in s, tellen we een fout bij k op.

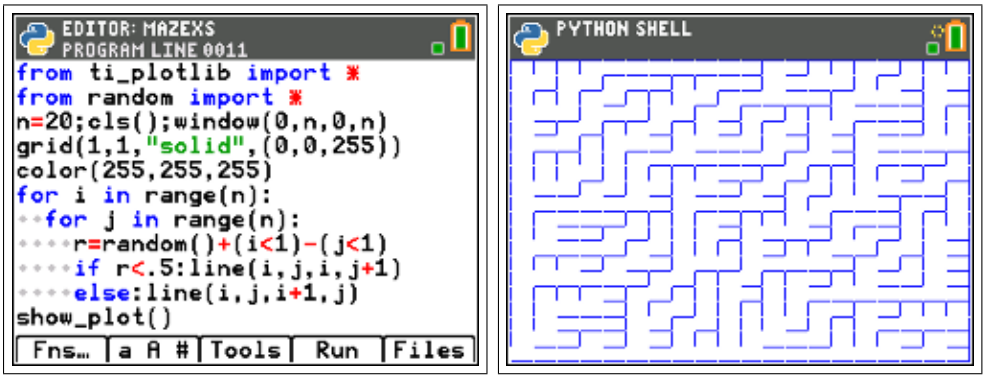
Je kan ook de eerste twee regels vervangen door een woordenboekje:

```
from random import *
s=list(choice(["sinus","functie","cosinus","tangens","parabool"]))
```

2.2.4 Doolhof

In dit programma gaan we een doolhof maken. Er bestaan veel algoritmen om dit te doen, de meeste echter zijn ingewikkeld en steunen op een meer grondige kennis van wiskunde (bijvoorbeeld grafentheorie). Onderstaande methode is echter zeer eenvoudig en produceert enkelvoudig samenhangende doolhoven, dit zijn doolhoven waavoor elke twee punten verbonden zijn door een unieke weg.

We beginnen met een raster van $n \times n$ cellen. Voor elke cel verwijderen we willkeurig ofwel de linkermuur ofwel de ondermuur. We letten er wel op dat in de eerste kolom en de onderste rij nooit een buitenmuur wordt weggehaald. Op deze wijze bekomen we een doohof. Vertrekken linksonder, moet je tot de rechterbovenhoek geraken.



- We importeren de nodige modules en gebruiken `from ti_plotlib import *` om de code kort te houden (je kan bij het ingeven de `plt`-versie ook gebruiken).

- We zorgen voor een grafisch venster met een blauw $n \times n$ raster en zetten daarna de kleur op wit, om muren weg te halen.

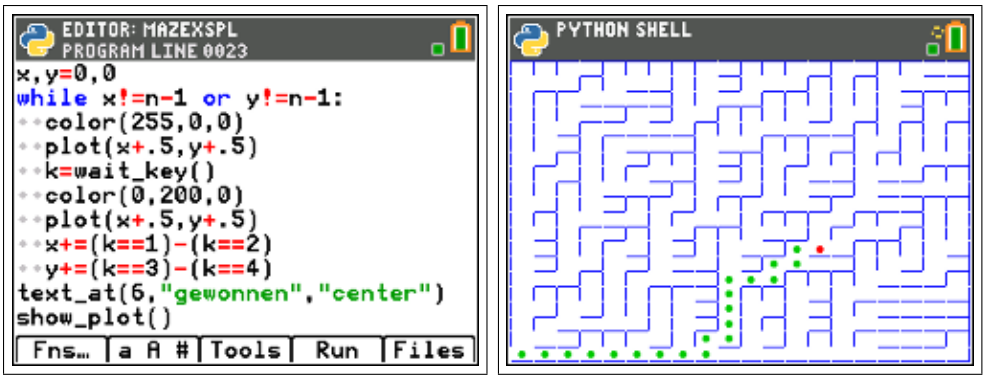
- Met twee lussen gaan we elke cel (i, j) af. Voor elke cel bepalen we een willekeurig getal `r=random()+(i<1)-(j<1)`. Dit getal zal zeker groter zijn dan $\frac{1}{2}$ als $i = 0$. Het zal negatief zijn, dus kleiner dan $\frac{1}{2}$, als $j = 0$. Hierme zullen we verzekeren dat in de eerste kolom en de onderste rij nooit een buitenmuur wordt weggehaald.

- Is $r < 0.5$ halen we de linkermuur weg, anders de ondermuur van de cel (i, j).

- Tenslotte tonen we met `show_plot()` het grafisch scherm.

2.2.5 Cursor door een doolhof

We zullen nu een cursor over de doolhof plaatsen zodat het spel gespeeld kan worden. Hiervoor moet je eerst `from ti_system import *` bovenaan toevoegen en `show_plot()` weghalen.



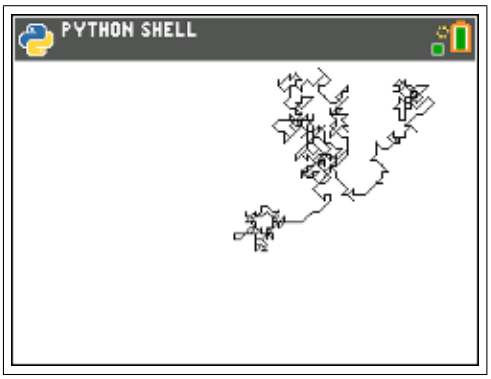
- De speler begint in de oorsprong.
- We starten een `while`-lus die duurt tot de speler in de rechterbovenhoek $(n-1, n-1)$ is geraakt.
- De huidige positie van de speler wordt in het rood aangeduid. Daarna wordt er gewacht tot een toets wordt ingedrukt.
- Als dit gebeurt, wordt de positie groen gekleurd en wordt de nieuwe positie bepaald. De pijltjestoetsen komen overeen met volgende codes $\rightarrow$: 1, $\leftarrow$: 2, $\uparrow$: 3, $\downarrow$: 4. Met `x+=(k==1)-(k==2)` zal dus x met 1 vergroot of verlaagt worden afhankelijk van welke toets werd ingedrukt, analoog bepaalt men ook de y-coördinaat. Daarna wordt de lus herhaald.
- Indien de lus stopt, heeft de speler zijn doel bereikt en wordt dit op het scherm getoont.

De doolhoven vormen een binaire boomstructuur en zijn eenvoudig op te lossen. Hoe doe je dit? Verklaar nu de naam binaire boom.

De speler kan in deze versie gerust doorheen de muren, er is geen wall-checking algoritme. Dit kan gedaan worden maar is een stukje moeilijker en vertraagt het programma. Je kan het spel uitdagender maken door `-(k==2)` en `-(k==4)` te verwijderen zodat een speler niet terug kan keren op zijn stappen.

2.2.6 Random walks

We bekijken het wandelpad van een dronkenman. Die is volledig bezopen en staat in het midden van een plein. Hij wandelt verder en telkens zet hij een stap in een willekeurige richting. De wandeling die hij aflegt noemt men in een *random walk*. Met volgend programma simuleren we deze wandeling van een dronkaard op de **TI-84+** .



Beschrijf in welke richtingen de dronkeman in het programma stappen kan zetten. Wat gebeurt er als je de code verandert in:

In het tweedimensionale geval geldt dat een random walk uiteindelijk terugkomt langs haar vertrekpunt:

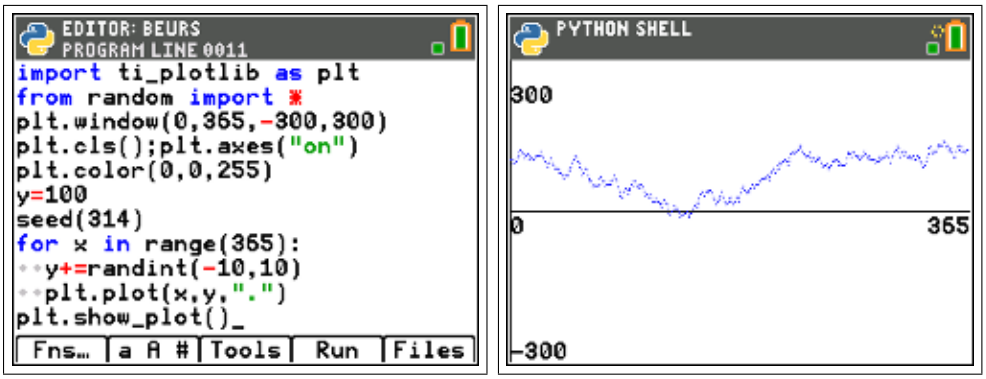
Een dronkeman die vanaf zijn huis vertrekt, zal uiteindelijk vroeg of laat toch thuis geraken.

In het driedimensionale geval echter, is het niet zeker dat een random walk terugkomt. Vaak wordt dit vertaald als:

Een mens mag dronken worden, maar een vogel niet.

2.2.7 Beurskoersen

Waar random walks een grote rol spelen is de beurs. Het aandeel van een bedrijf is op een bepaalde dag 100 euro waard. Veronderstel dat, onder normale omstandigheden, de aandeelprijs dagelijks met maximaal met 10 euro kan stijgen of dalen. Wanneer er (te) grote fluctuaties optreden wordt de beurs immers stilgelegd. De grafiek van de bekomen koers gedurende een jaar is een random walk. Het programma dat zulke grafieken maakt ziet er als volgt uit.



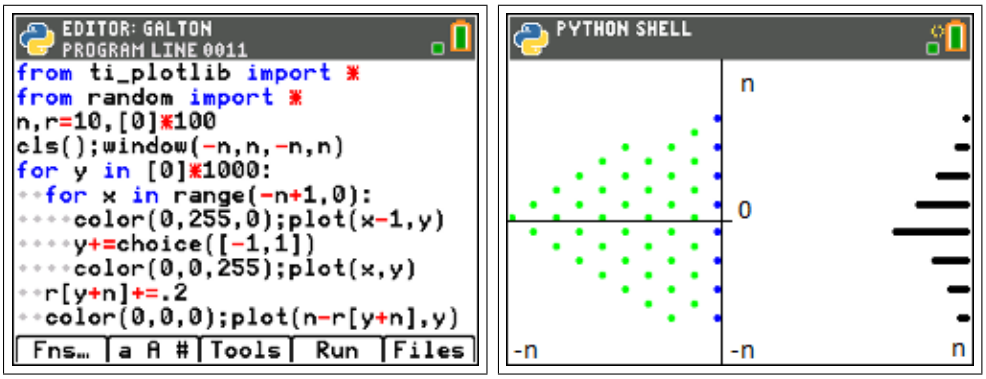
- We laden de nodige modules en stellen een grafisch venster in met een assenkruis, zodat de x-as een volledig jaar spant. De startwaarde van ons aandeel is $y = 100$.
- De module `random` bezit een aantal functies die een rij willekeurige getallen genereren. De startwaarde van deze rijen noemt men de *seed*. Wanneer `seed()` niet wordt aangeroepen, start de rij met als eerste waarde een getal afgeleid van de tijd sinds men het toestel heeft aangezet in microseconden. Wanneer je echter een waarde aan `seed()` meegeeft, zal de randomgenerator starten met deze waarde en krijg je dus altijd dezelfde reeks willekeurige getallen.
- Met een `for`-lus gaan we elke dag een zekere willekeurige winst of verlies bij de waarde van het aandeel optellen. Dit gebeurt met `y+=randint(-10,10)`. Waarna we de nieuwe waarde van het aandeel op de grafiek zetten.

Laat het programma een aantal keer draaien met verschillende *seeds*. Je krijgt telkens een andere koers te zien. Vind er eentje die stijgt, eentje die daalt en eentje waar de waarde van het aandeel na een jaar niet veel veranderd is. Geef telkens de bijhorende seed. Men kan aantonen dat elke ééndimensionale random walk uiteindelijk toch op nul zal komen.

Op de beurs zal je uiteindelijk verliezen. Hoe groter je startkapitaal, hoe langer het zal duren.

2.2.8 Galton bord

Een Galton bord is een verticale plank met spijkers in een geschrankt patroon. Midden bovenaan laat men knikkers rollen die op de spijkers vallen en telkens ofwel naar links of naar rechts uitwijken. Een knikker volgt aldus een willekeurig pad tussen de spijkers. Onderaan worden de knikkers opgevangen in verschillende bakjes. Op deze wijze zien we de een normaalverdeling verschijnen. Volgend programma simuleert een bord van Galton. We gebruiken ditmaal het scherm in landscape.



- We importeren de nodige modules met `from ... import *` zodat de commando's korter zijn en het programma op één scherm past (zie opm. 9). Vervolgens stellen we n rijen spijkers in. r stelt 100 lege bakjes voor, waar de knikkers in zullen eindigen (`[0]*4=[0,0,0,0]`).

- Het grafisch venster (zie figuur) heeft assen van $-n$ tot n. Op het interval $[-n, 0]$ van de x-as simuleren we het vallen van de knikkers, terwijl aan de rechterzijde op $[0, n]$ de uitkomsten cumulatief getekend worden.

- Met `for y in [0]*1000` laten we duizend knikkers vanuit hoogte $y = 0$ vallen.

- De volgende `for`-lus simuleert de val van één knikker. Elke keer dat die een spijker tegenkomt wordt de nieuwe positie bepaald met `y+=choice([-1,1])` nadat de oude positie (het spoor) in het groen werd getekend. Nadien tekenen we de nieuwe positie in het blauw, zodat je de knikkers kan zien vallen.

- Tenslotte wordt de laatste positie y bijgeteld in het bakje `r[y+n]` en wordt dit in het zwart getekend. We gebruiken hierbij wel gewicht van 0.2 om grafisch een beter resultaat te krijgen. Daarna is het de beurt aan de volgende knikker.

2.2.9 Binaire getallen

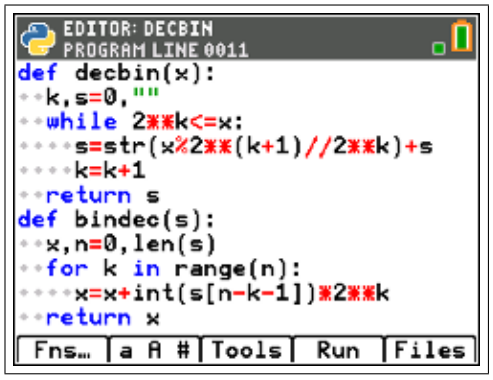
Een natuurlijk getal noteren we traditioneel in het tientallig of decimaal stelsel met grondtal 10. Dit betekent dat getallen worden genoteerd met cijfers uit $\{0, 1, 2, \ldots 9\}$ die telkens een veelvoud van een macht van tien voorstellen:

$$1273 = 1 \cdot 10^3 + 2 \cdot 10^2 + 7 \cdot 10^1 + 3 \cdot 10^0$$

Computers gebruiken intern een ander talstelsel, namelijk het binaire stelsel. Dit is een talstelsel op grondtal 2, elk getal bestaat dan uit de cijfers $\{0, 1\}$ die een veelvoud van een macht van 2 voorstellen:

$$101011 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Dit is dus het getal 43 in decimale notatie.
Onderstaand procedures zet getallen om tussen beide talstelsels.



Merk op dat we hier binaire getallen schrijven als een textstring.

2.2.10 Getallen in een andere basis.

Met onderstaande procedure kan je getallen van de ene basis naar de andere omzetten.

- Met de eerste `for`-lus zetten we de string `sx` om naar een decimale getalwaarde x.
- Met de tweede `for`-lus wordt nagegaan hoeveel keer de macht b^k van de basis b past in de rest van x na deling door b^{k+1}. Dit geeft het $k$de cijfer van x in basis b.

Ga manueel na hoe het algoritme werkt.

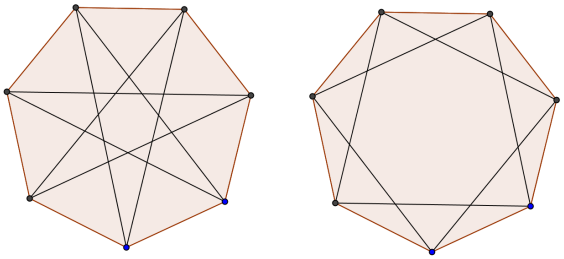
2.2.11 Caesar-code

Reeds in de oudheid was het van belang om boodschappen door te sturen zonder dat de vijand de boodschap kon begrijpen indien deze toch onderschept werd. Julius Caesar maakte veelvuldig gebruik van de naar hem genoemde Caesar-code. Om een bericht te encoderen werd vooraf een sleutel afgesproken tussen Caesar en de bestemmeling. Deze sleutel k was een getal van 1 t.e.m. 25. Een bericht werd dan opgeschreven, waarna elke letter k plaatsen in het alfabet verschoven werd. Om het bericht te decoderen moest men gewoon elke letter van het alfabet in de andere zin verschuiven. We kunnen nu de **TI-84+** gebruiken om de Caesar-code te programmeren. Het programma ziet er als volgt uit.

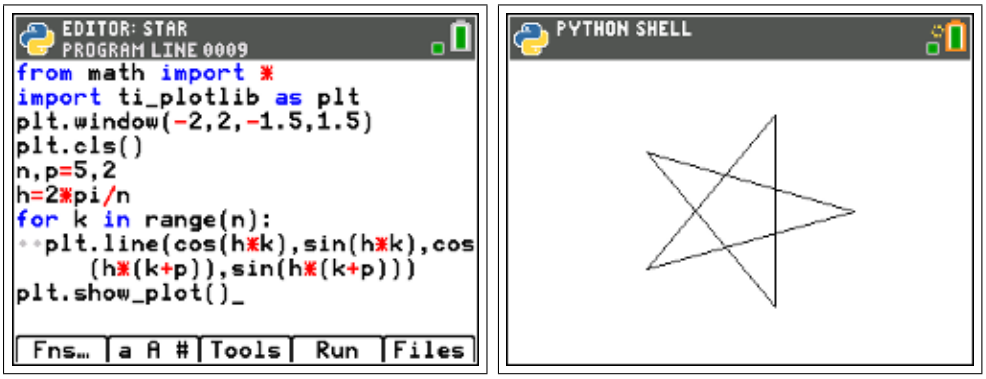
- We beginnen met het alfabet in te geven in `az`.
- De procedure `encode(s,k)` gaat ervan uit dat de boodschap in s zit en de sleutel k is. Het resultaat r is aanvankelijk leeg.
- Voor elk letterteken c uit s wordt met `az.index(c)` de positie in het alfabet bepaald, waarna geëncodeerde letter wordt bekomen door de positie `(az.index(c)+k)%26`. Deze wordt toegevoegd aan r.
- Indien c geen letter is, wordt het ongecodeerd aan r gevoegd.
- Uiteindelijk geeft de procedure het gecodeerde resultaat r terug.
- De procedure `decode(s,k)` verschuift het alfabet gewoon over $-k$ om te decoderen.

2.2.12 Sterren

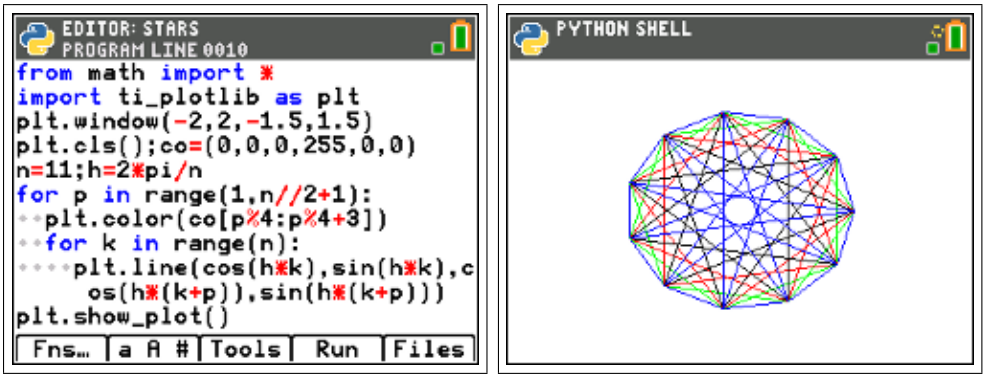
Een $\{n|p\}$-ster is een vlakke figuur bekomen door de punten van een regelmatige n-hoek telkens te verbinden met het $p$de volgend punt.



Onderstaand programma tekent een $\{n|p\}$-sterren voor gegeven n en p.



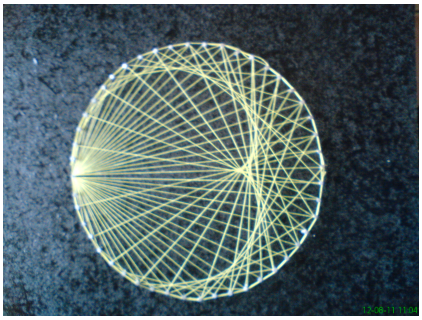
Met enkele kleine aanpassingen kan je alle sterren tekenen voor een gegeven n.



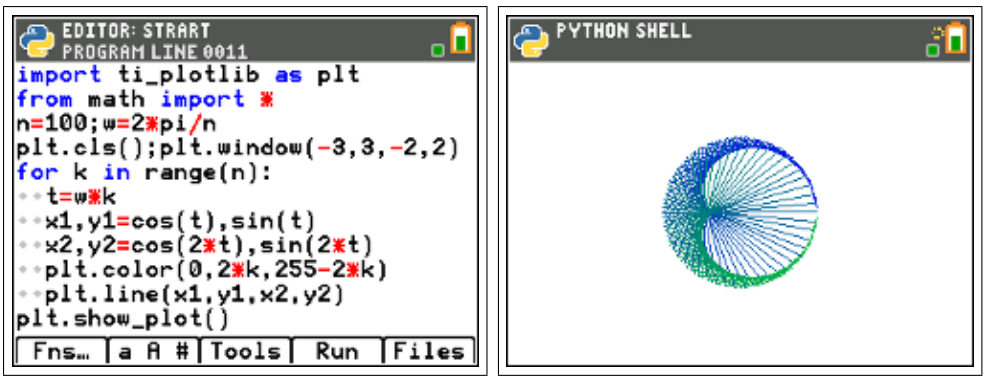
We gebruiken hier het programmeertrucje van opmerking 10 over verkorte kleurenpaletten om de kleuren te selecteren.

2.2.13 String art

In de 'string art' worden vormen bekomen door touwtjes tussen spijkers te spannen. We maken het voorbeeld van een cardioïde. We verdelen onze spijkers gelijkmatig over een cirkel en verbinden de $t$de spijker met de $2t$de, i.e. het punt $(\cos t, \sin t)$ met $(\cos 2t, \sin 2t)$. Aldus bekomen we volgend resultaat.



Met onderstaand programma wordt telkens het lijnstuk getrokken van (x_1, y_1) naar (x_2, y_2). De plaats van de punten/spijkers kunnen we aanpassen.



Met `plt.color(0n,2*k,255-2*k)` zorgen we ervoor dat de kleur van de touwtjes langzaam van blauw naar groen overgaat. Door na de `plt.line()` de onderstaande opdrachten toe te voegen, kan je de spijkers ook tekenen.

```
plt.color(255,0,0)
plt.plot(x1,y1,"o")
plt.plot(x2,y2,"o")
```

Je kan volgende voorbeelden uitwerken.

- $(x_1, y_1) = (\cos t, \sin t)$ en $(x_2, y_2) = (\cos(t + \phi), \sin(t + \phi))$
- $(x_1, y_1) = (\cos 2t, \sin 2t)$ en $(x_2, y_2) = (\cos 3t, \sin 3t)$
- $(x_1, y_1) = (t, 0)$ en $(x_2, y_2) = (0, 10 - t)$ met $0 < t < 10$

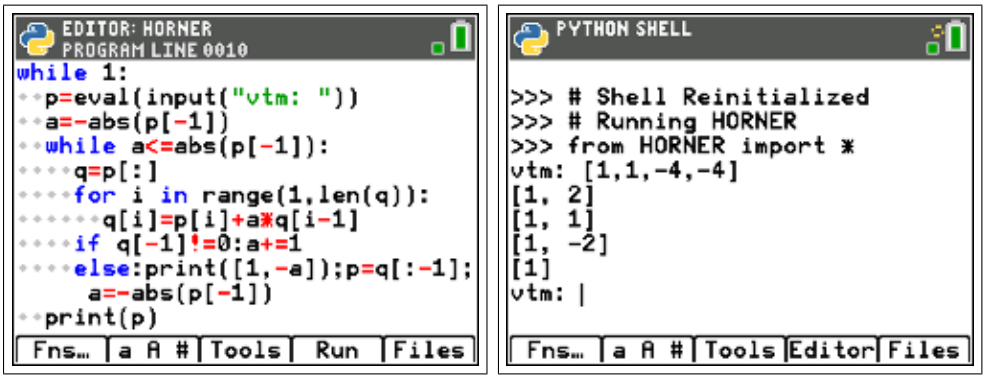
Zoek ook eens naar 'string art' in Google Afbeeldingen.

2.2.14 Horner

het algoritme van Horner laat toe om een veelterm $P(x)$ te ontbinden in factoren. Het schema om de deling uit te voeren ziet er als volgt uit:

	$\ldots$	$\ldots$	$\ldots$	$\ldots$		$\ldots$	$\leftarrow$	$P(x)$
a		$\ldots$	$\ldots$	$\ldots$		$\ldots$	$\leftarrow$	$+a \cdot Q(x)$
$Q(x) \rightarrow$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\mid$	$\ldots$	$\leftarrow$	R

Indien $R = 0$, dan is de veelterm deelbaar door $(x - a)$ en bekomt men de ontbinding $P(x) = (x - a) \cdot Q(x)$. Herhaaldelijk toepassen van dit schema voor alle gehele delers van de constante term geeft een volledige ontbinding indien de veelterm enkel gehele nulwaarden heeft.



- We starten met een oneindige lus. Telkens wordt de gebruiker om een veelterm gevraagd die als een lijst wordt ingegeven: `[1,1,-4,-4]` staat dus voor de veelterm $x^3 + x^2 - 4x + 4$. Met `eval` wordt dit omgezet naar een Pyhton lijst `p`.

- Met `p[-1]` nemen we het laatste element van de lijst `p`, dit is de constante term c van de veelterm. We starten met kandidaat Hornergetal $a = -|c|$. We zullen niet de moeite doen om de delers van c te zoeken, maar gaan gewoon alle waarden van $-|c|$ tot $|c|$ met een `while`-lus af.

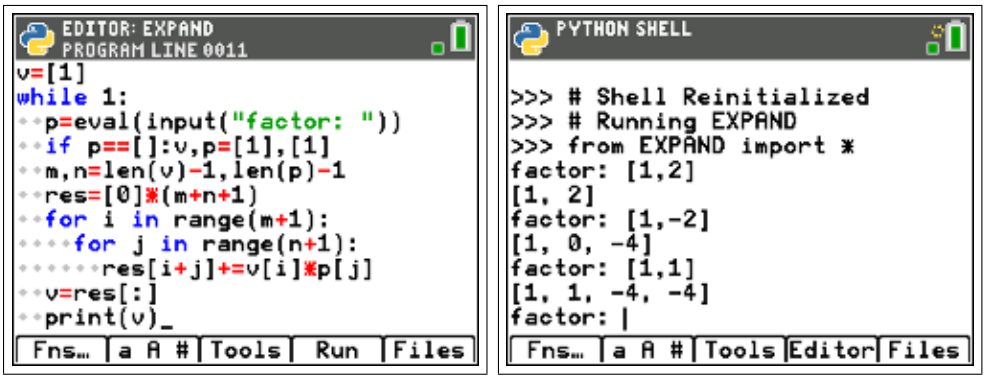
- Een `for`-lus bepaalt de coëfficiënten van $Q(x)$: `q[i]=p[i]+a*q[i-1]`.

- Het laatste element van `q` is de rest R. Indien deze niet nul is, proberen we een volgend Hornergetal a. Anders doen we drie dingen. De factor $(x - a)$ afdrukken, $P(x)$ gelijkstellen aan $Q(x)$ (zodat het algoritme verder kan) en a aanpassen aan de nieuwe constante term.

- Wanneer de `while`-lus afgewerkt is en er geen verdere deler $(x-a)$ werd gevonden, zal het programma het de overblijvende factor afdrukken, daarna start het opnieuw met het vragen naar een veelterm.

2.2.15 Veeltermproducten

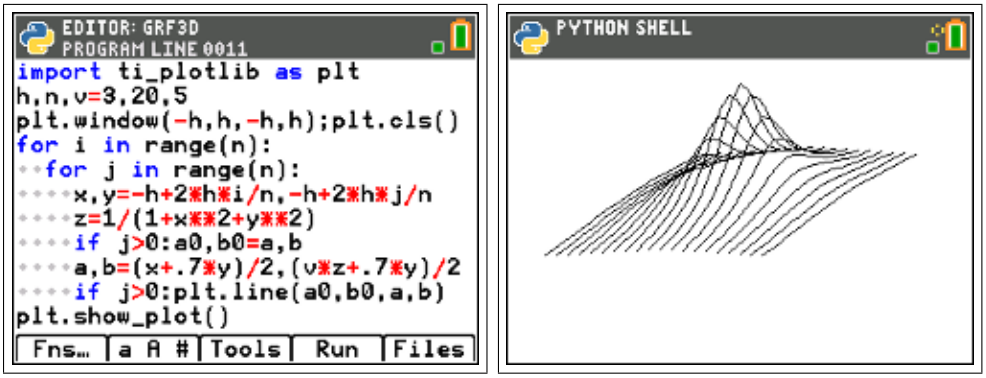
We kunnen de **TI-84+** leren om symbolisch met veeltermen te rekenen. In dit voorbeeld zullen we het veeltermproduct van verschillende factoren uitrekenen.



- De veelterm `v` houdt het huidige product bij. Aanvankelijk is dit 1.
- Met de oneindige lus `while 1` blijven we factoren opvragen aan de gebruiker. Met `p=eval(input())` wordt een lijst, opgegeven door de gebruiker omgezet in een Python lijst. De invoer `[1,2]` staat voor de factor $(x+2)$. Indien de gebruiker een lege lijst `[]` zou invoeren, wordt het huidige product terug op 1 gezet.
- Hierna worden de graden m en n bepaald van het huidige product v en van de nieuwe factor p. Het tijdelijke resultaat `res` is een veelterm van graad $m+n$, dus een lijst van lengte $m+n+1$, die aanvankelijk enkel nullen bevat: `[0]*(m+n+1)`
- Met twee `for`-lussen gaan we alle termen van beide factoren v en p distribueren. Telkens wordt de bekomen term `v[i]*p[j]` (van graad $i+j$) opgeteld in `res[i+j]`.
- Tenslotte wordt het huidig product v het pas berekende resultaat `res` en wordt dit afgedrukt. Waarna een nieuwe factor wordt opgevraagd.

2.2.16 Grafieken van functies in 3d

Onderstaand programma maakt de grafiek van een functie $z = f(x, y)$ in 3d.

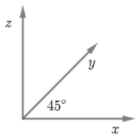


- We importeren `ti_plotlib` en gebruiken een $n \times n$ grid om in het xy-vlak het vierkant $[-h, h]\times[-h, h]$ onder te verdelen. v stelt een verticale schaalfactor voor waarmee we de hoogte van de grafiek kunnen aanpassen. Voor het 2d grafisch scherm gebruiken we `plt.window(-h,h,-h,h)`.

- Met twee lussen gaan we alle punten `x,y=-h+2*h*i/n,-h+2*h*j/n` van het grid af. Telkens berekenen we de functiewaarde z (hier $z = \frac{1}{1+x^2+y^2}$).

- Indien het niet gaat om een beginpunt in de y-richting (dwz $j > 0$), dan onthouden we in (a_0, b_0) het laatst getekend punt.

- We berekenen de 2d projectie (a, b) van het nieuwe punt (x, y, z) d.m.v. een eenvoudige cavalierperspectief onder een hoek van 45°. Het scherm valt samen met het xz-vlak, een punt op een diepte y wordt door de projectie verschoven in de 45°-richting.



We bekomen:

$$\begin{cases} a &= x + \frac{\sqrt{2}}{2}y \\ b &= z + \frac{\sqrt{2}}{2}y \end{cases}$$

We passen de schaalfactor v toe op z en een homothetie met factor $\frac{1}{2}$ op de 2d projectie, zodat de punten binnen `plt.window(-h,h,-h,h)` vallen.

- We tekenen het lijnstuk van (a_0, b_0) naar (a, b), tenzij het om een beginpunt gaat. Daarna gaan we naar het volgend punt van het grid.

Door `plt.color(0,150,255*(z<0))` toe te voegen voor `plt.line()` worden punten onder het xy-vlak in het blauw getekend, die erboven in het groen.

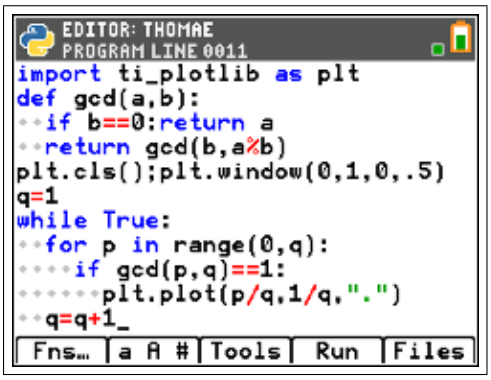
2.2.17 Thomae's functie

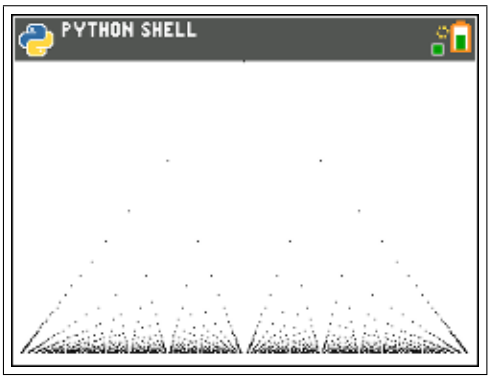
De functie van Thomae is wordt gegeven door:

$$f(x) = \begin{cases} \frac{1}{q} & x = \frac{p}{q} \in \mathbb{Q} \text{ onvereenvoudigbaar} \\ \\ 0 & x \in \mathbb{R}\backslash\mathbb{Q} \end{cases}$$

Deze functie is een voorbeeld van een functie discontinu is in elk punt van $\mathbb{Q}$, maar continu is op $\mathbb{R}\backslash\mathbb{Q}$. Verder is het een periodieke functie met periode 1 en is ze nergens afleidbaar. Bovendien heeft ze in elk punt van $\mathbb{Q}$ een lokaal maximum.

De grafiek maken van deze functie vergt een apart programma maar geeft wel een mooi resultaat.





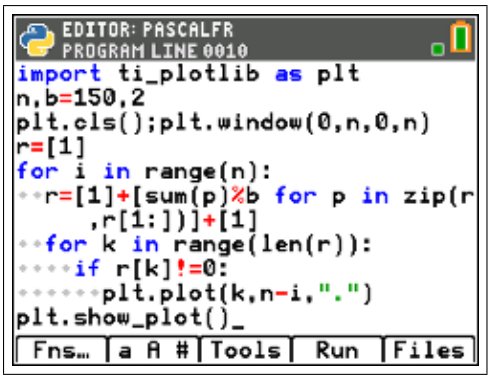
- Aanvankelijk definiëren we de procedure `gdc(a,b)` die de grootste gemene deler van a en b bepaald d.m.v. recursie.
- Nadien wordt het grafisch scherm leeggemaakt en wordt de noemer q aanvankelijk 1. Met een `for`-lus gaan we alle mogelijke tellers van de breuken p/q af.
- Indien de breuk onvereenvoudigbaar is (de grootste gemene deler is dan 1), wordt het beeldpunt getekend. De parameter `"."` zorgt voor een dun puntje.
- Tenslotte nemen we de volgende noemer en beginnen we opnieuw.

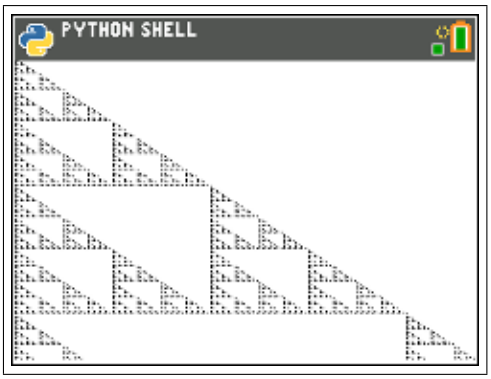
Hiermee kunnen we de functie tekenen op het interval $[0, 1[$. Om de periodiciteit na te gaan kan je de lus aanpassen zodat p gaat tot $3q$. Je bekomt dan de grafiek op het interval $[0, 3[$. Door kleine aanpassingen kan je ook volgende functie tekenen, deze is overal discontinu behalve in 0.

$$f(x) = \begin{cases} x^2 & x = \frac{p}{q} \in \mathbb{Q} \text{ onvereenvoudigbaar} \\ \\ 0 & x \in \mathbb{R}\backslash\mathbb{Q} \end{cases}$$

2.2.18 De driehoek van Pascal en de zeef van Sierpinski

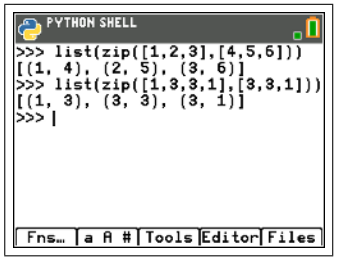
We maken een programma dat de driehoek van Pascal tekent. Hierbij tekenen we enkel de oneven getallen, de even getallen worden blanco gelaten. Het programma ziet er als volgt uit:





- We importeren de grafische module en stellen het aantal rijen n in. Met $b = 2$ zullen we ervoor zorgen dat de even getallen niet worden getekend. Daarna stellen we het grafisch venster in. De eerste rij van de driehoek is `r=[1]`. Met een `for`-lus gaan we de opeenvolgende rijen berekenen en tekenen.

- Met `zip()` kan je koppels maken van overeenstemmende elementen uit twee lijsten.



  `[sum(p)%b for p in zip(r,r[1:])` doet het volgende: eerst worden met de vorige rij `r` en de vorige rij `r[1:]` zonder het eerste getal, koppels gemaakt. Deze worden opgeteld modulo b (we willen enkel weten of de getallen even of oneven zijn). Tenslotte voegen we het eerste en het laatste element toe. We bekomen aldus de volgende rij uit de driehoek.

- Met een volgende `for`-lus gaan we de nieuwe rij af en tekenen we enkel een punt indien het getal niet even is.

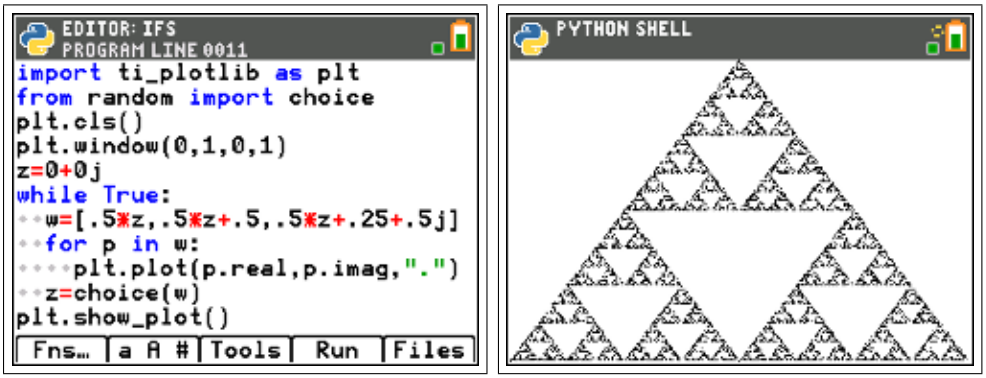
De bekomen fractaal noemt men de *zeef van Sierpinski*. Wat bekom je als je al de drievouden blanco laat? En de vijfvouden? Wat loopt er mis bij de zesvouden? Kan je dit verklaren?

2.2.19 IFS

Een IFS (Iterated Function System) bestaat uit een aantal transformaties van het vlak. Men kiest een willekeurig startpunt (meestal de oorsprong) en past dan deze transformaties toe. Men tekent de punten en kiest willekeurig één ervan als nieuw startpunt, waarna men de procedure herhaald. Het resultaat is een figuur die fractale eigenschappen heeft (zelfgelijkvormigheid).

In onderstaand programma worden de transformaties beschreven in het complexe vlak.

1. homothetie: $z \mapsto 0,5z$
2. homothetie met verschuiving: $z \mapsto 0,5z + 0,5$
3. homothetie met verschuiving: $z \mapsto 0,5z + (0,25 + 0,5i)$



Kan je ook draaiingen implementeren?
Kijk ook eens onder 'iterated function systems' op Google Afbeeldingen.

2.2.20 Julia-fractalen: Backtracking methode

Een Julia-fractaal is de verzameling van alle complexe getallen z_0 waarvoor de recursieve rij $z_n = z_{n-1}^2 + c$ convergeert. Hierbij zal de fractaal afhangen van de parameter c.

De fractalen van de Franse wiskundige Gaston Julia vergen zeer veel rekenwerk. Elk punt van het scherm moet immers dienen als startwaarde van de rij. Men moet dan een minstens een viftigtal termen uit de rij berekenen om te achterhalen of ze convergeert of niet. Dit kan gemakkelijk op een computer, maar de **TI-84+** is hiervoor te traag.

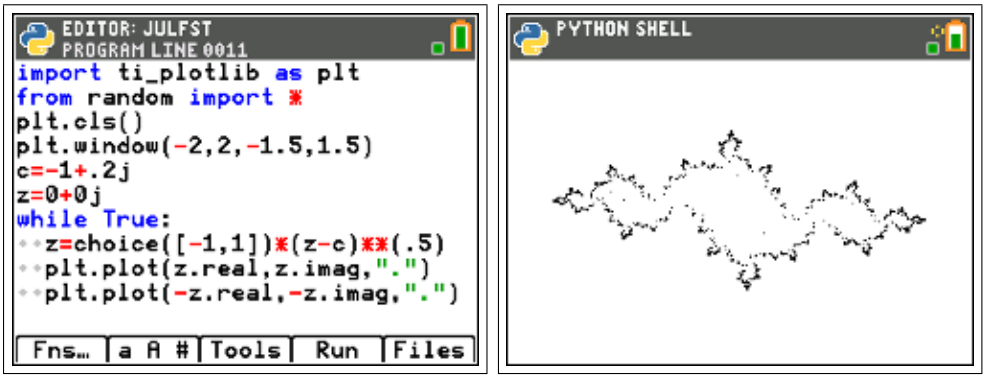
Gelukkig bestaat er een andere methode: *de backtracking methode*. We beschouwen opnieuw de rij

$$z_n = z_{n-1}^2 + c$$

Tot nu toe hebben we telkens we een term kenden, de volgende term uitgerekend met deze formule. Maar stel dat we een punt uit de rij kennen, dan kunnen we de vorige berekenen (dit heet *backtracking*).

$$z_{n-1} = \pm\sqrt{z_n - c}$$

Men kan bewijzen dat je voor een Julia fractaal mag beginnen met de oorsprong als startwaarde. Als je telkens willekeurig kiest tussen beide wortels uit de backtracking formule, dan zal de rij die je bekomt een rij punten zijn uit de rand van de fractaal. Als je dit lang genoeg doet bekom je een volledig beeld van de rand van de fractaal. Op de **TI-84+** ziet het programma er als volgt uit.



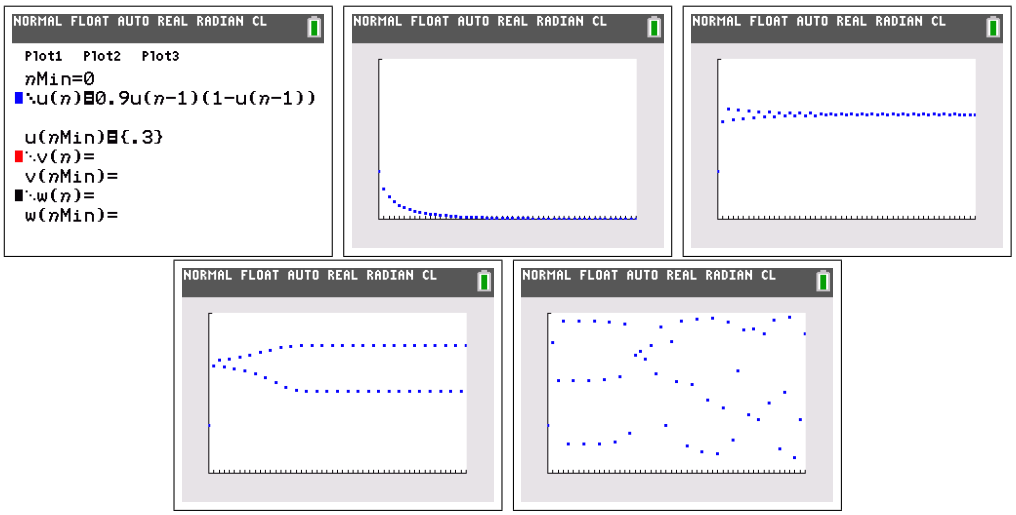
Maak de Julia fractalen voor volgende waarden van c. Verklaar hun naam.

- $c = i$ (dendriet)
- $c = -\frac{3}{4}$ (San Marco fractaal)
- $c = 1$ (stofwolk)

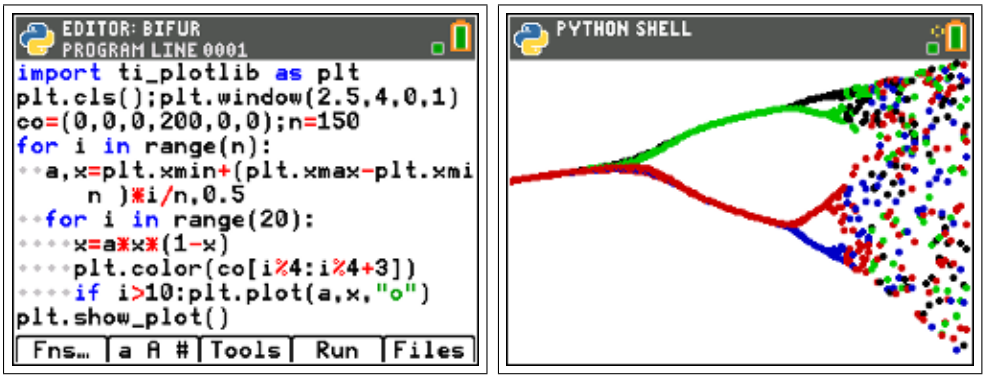
Zoek zeker eens Julia-fractalen op via Google Afbeeldingen.

2.2.21 Model van Verhulst en chaos

De recursieve rij $u_n = au_{n-1}(1 - u_{n-1})$ is in de biologie bekend als "het model van Verhulst". Voor verschillende waarden van $0 < a < 4$ verandert het convergentiegedrag volledig. Eerst convergentie, daarna adherentie en tenslotte zelfs een chaotisch gedrag. We kunnen dit grafisch aantonen.



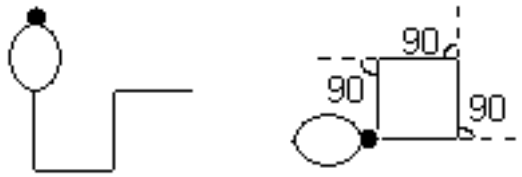
Om dit convergentiegedrag in kaart te brengen kunnen we een bifurcatiediagram maken. We plaatsen op de x-as de parameter a. Voor elke waarde van a tekenen we de punten van de rij, na een stabilisatiefase van 10 termen. Indien de rij adhereert aan twee waarden, zullen dus twee punten getekend worden boven de waarde van a.



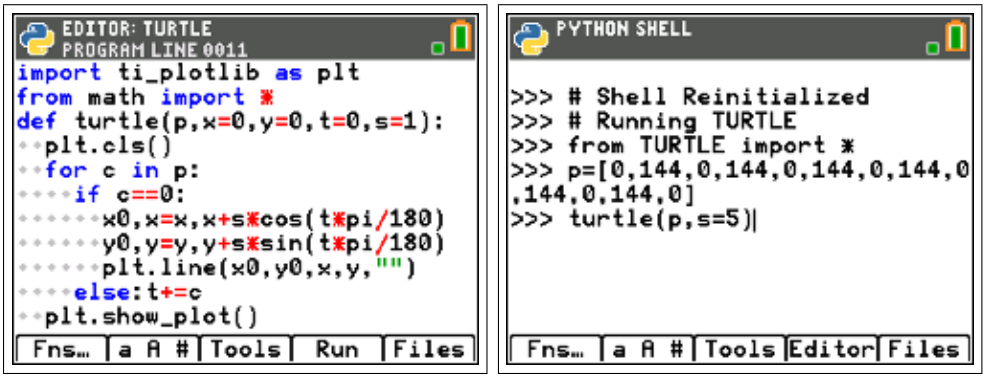
Opeenvolgende punten van een rij krijgen telkens een andere kleur dankzij het programmeertrucje van opmerking 10. We zien duidelijk het chaotisch gedrag verschijnen. De eerste bifurcatie heeft plaats rond $a = 2,8$. Elke tak splitst opnieuw tot het systeem volledig chaotisch wordt. Midden in het chaotisch gedeelte is er plots weer structuur. We zien dit duidelijk als we inzoomen (`plt.window(3.8,4,0,1)`).

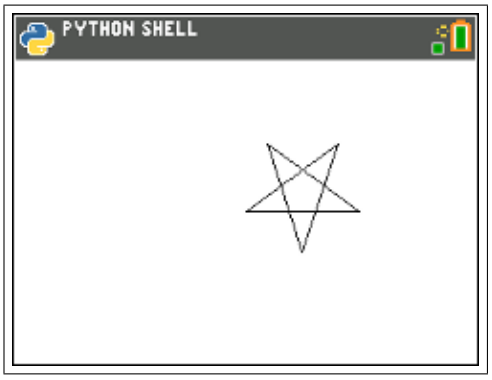
2.2.22 Turtle graphics: een eigen programmeertaal maken!

We beschouwen een schildpad in een veld. Het diertje verplaatst zich telkens over een welbepaalde lengte. Ze kan zich ook draaien over een bepaalde hoek. Op die manier laat ze een spoor achter. Je kan de schildpad nu 'programmeren' door te zeggen wanneer ze vooruit moet (0) en wanneer ze moet draaien (getal, vb. 90°). We kunnen nu een vierkant tekenen door de opdracht: $0, 90, 0, 90, 0, 90, 0$



De afspraken die we hebben gemaakt leggen een eenvoudige programmeertaal vast, het programma om een vierkant te tekenen bestaat uit de bovenstaande opdrachten. We zullen dit implementeren in Python. De schildpad start in $(x, y) = (0, 0)$, met haar neus in de positieve x-richting ($t = 0°$). De stapgrootte is s. Het programma is een lijst getallen (p).





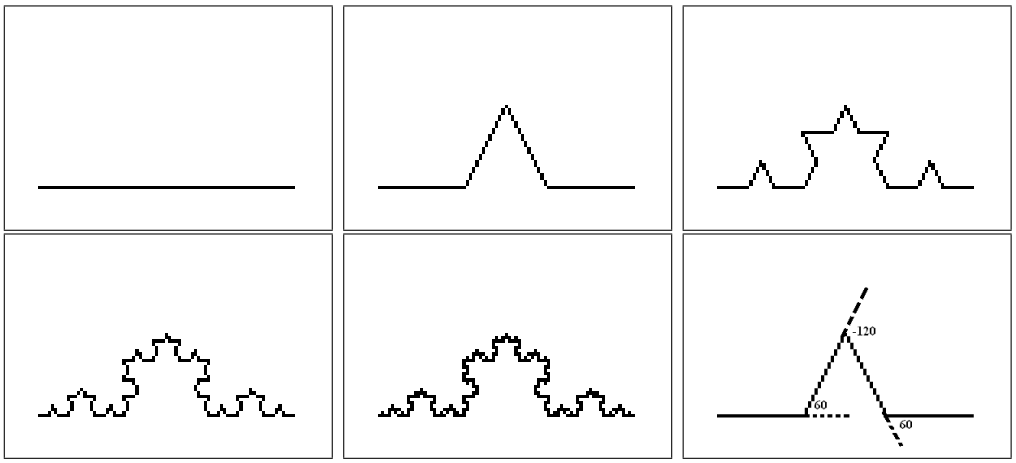
Merk op dat we met `turtle(p,s=5)` de parameter s op naam ingeven, dit staat toe om de standaardwaarden voor de andere parameters te gebruiken.

Enkele mogelijke opdrachten:

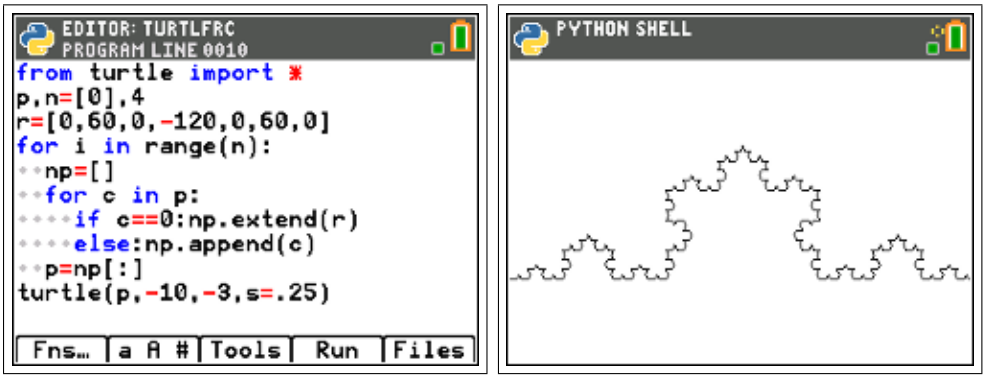
- Gebruik je schildpad-programma om een gelijkzijdige driehoek en een rechthoek te tekenen.
- Met welk schildpad-programma kan je een rechthoekige driehoek tekenen? Welke stelling(en) heb je hiervoor nodig?
- Schrijf een schildpad-programma om een mannetje te tekenen.

2.2.23 De Koch kromme

We zullen nu onze schildpad-programmeertaal gebruiken om fractalen te maken. We vertrekken van een lijnstuk. Dit wordt vervangen door een bepaald patroon. Elk lijnstukje uit dit patroon wordt opnieuw vervangen door een verkleinde versie van dit patroon. Dit procédé blijft men herhalen. De uiteindelijke figuur is een kronkelige fractaal.



De bovenstaande fractaal is de *Koch kromme*. Het herhaalde patroon is gegeven door: $0, 60, 0, -120, 0, 60, 0$. Het programma ziet er als volgt uit.

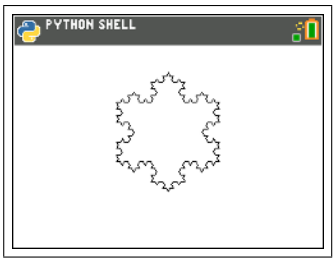


- We importeren eerst onze module `turtle` en stellen de beginfiguur p in als een lijnstuk. We zullen 4 keer het patroon r substitueren.
- Met een lus starten we n iteraties. Telkens is de nieuwe figuur np aanvankelijk leeg. Voor elke c in de vorige figuur p zijn er twee mogelijkheden.
- Indien c nul is (een lijnstuk), wordt het patroon r toegevoegd aan de nieuwe figuur. Anders wordt enkel de hoek toegevoegd.

- Vóór de volgende iteratie wordt de inhoud van de nieuwe figuur np gekopieerd naar p.
- Tenslotte roepen we `turtle()` aan, waarbij we de figuur p tekenen, beginnend in het punt $(-10, -3)$ en stapgrootte $s = 0.25$. Dit geeft een mooi gecentreerd beeld.

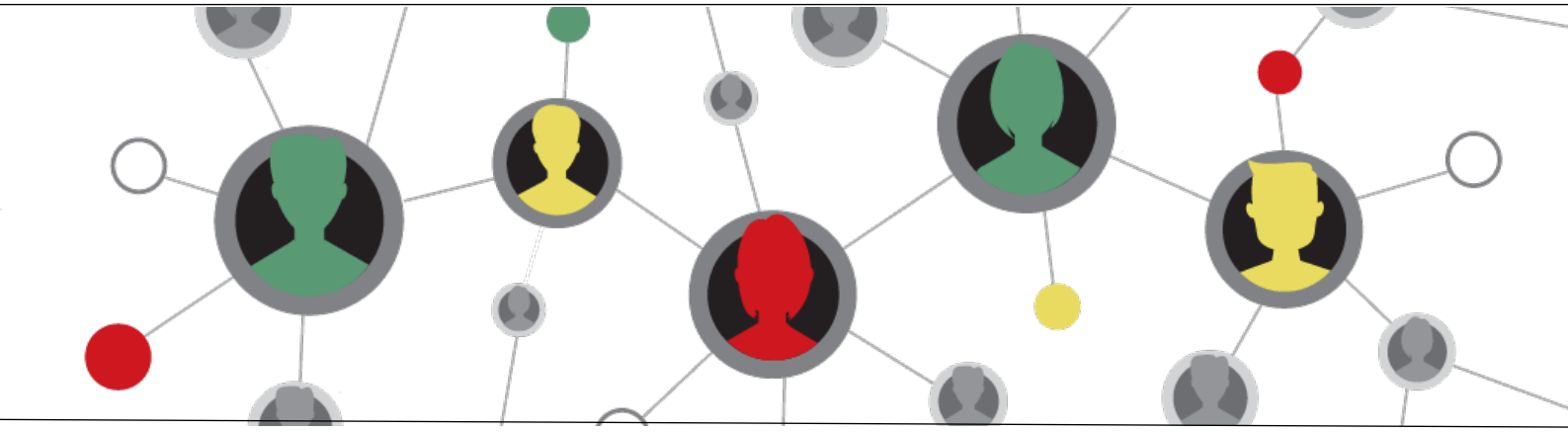
Als je niet begint met een lijnstuk maar met volgende figuur krijg je een Koch sneeuwvlok of Koch eiland.

$$120, 0, -120, 0, -120, 0$$



Welke fractaal bekom je als je vertrekt van een lijnstuk en volgend patroon gebruikt?

$$0, 120, 0, -120, 0, -120, 0, 120, 0$$



T³ NEDERLAND

T³ VLAANDEREN

Texas Instruments