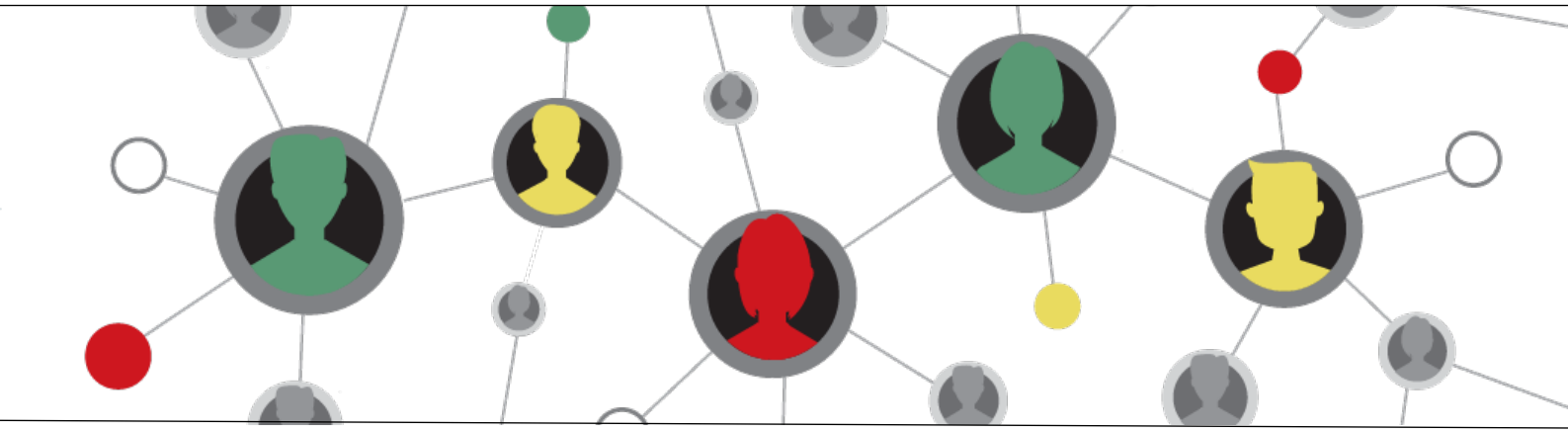
TI Python BootCamp

Python Basics



Bert Wikkerink – Koen Stulens

T³ NEDERLAND

Teachers Teaching with Technology™

T³ VLAANDEREN

TI Python BootCamp

Python Basics

Deel 1 Variabelen, berekeningen & beslissingen

Deel 2 Iteraties & Functies

Deel 3 Objectgeoriënteerd programmeren

1. Wat is een variabele?

van Dale

varia'bel in staat om van getal, afmeting, vorm, plaats, enz. te wisselen, resp. veranderd te worden

varia'bele (wisk) grootheid die in waarde enz. kan wisselen

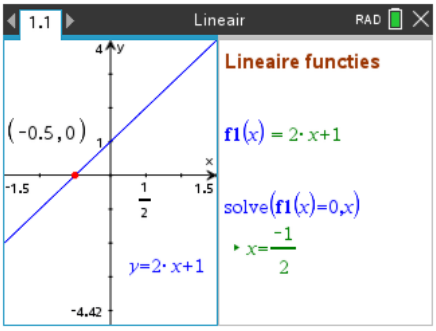
WIKIPEDIA
De vrije encyclopedie

nl.wikipedia.org/wiki/Variabele (wiskunde)
Een variabele is de aanduiding van een willekeurig element van een verzameling. De variabele neemt waarden aan in die verzameling. Een variabele wordt meestal voorgesteld door een letter.

nl.wikipedia.org/wiki/Variabele (informatica)
Een variabele is een term die gebruikt wordt in verband met programmeren. In de code van een computerprogramma associeert een variabele een naam met een of meer geheugenadressen.

Voor reële functies $f:\mathbb{R} \longrightarrow \mathbb{R}:x \longmapsto 2x+1$ noemen we x de variabele en $f(x)=2x+1$ het beeld van x. $f(x)$ wordt ook genoteerd d.m.v. y: $y=f(x)$ of $y=2x+1$ of $y(x)=2x+1$.

Hierbij wordt x de onafhankelijke variabele genoemd en $y=2x+1$ de afhankelijke variabele, waarbij het lineaire verband $y=2x+1$ aangeeft hoe y verandert i.f.v. x. $y=2x+1$ is de vergelijking van de grafiek van de functie f.

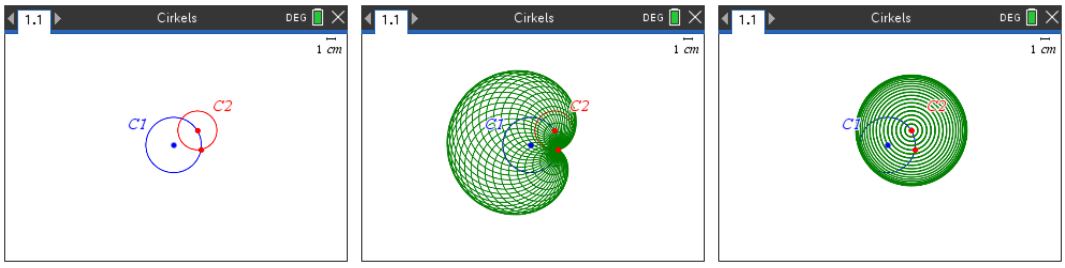


Het bepalen van de nulpunten van $f(x)=2x+1$ (snijpunten x-as) komt neer op het oplossen van de vergelijking $y=0 \Leftrightarrow 2x+1=0$ en omgekeerd.

Wat als we de rol van x en y omdraaien, $x(y)$?

$$y(x)=2x+1 \Leftrightarrow x(y)=\frac{1}{2}y-\frac{1}{2}$$

Onafhankelijke en afhankelijke variabelen kunnen goed geïllustreerd worden a.h.v meetkundige construties. Bijvoorbeeld, teken een cirkel *C1* (onafhankelijk) en een cirkel *C2* (afhankelijk) met middelpunt en radiuspunt op de cirkel *C1*. Voor de cirkel *C1* kan het middelpunt willekeuring verplaats worden als ook de straal verandert. Het veranderen van de cirkel *C2* is beperkt tot het verplaatsen van middel- en radiuspunt op de cirkel *C1*. Hieronder de meetkundige plaatsen bepaalt door de cirkel *C2* met als eerst het variabele middelpunt en dan het radiuspunt.



En nog een andere visualisatie van onafhankelijke en afhankelijke variabelen.

Onafhankelijke variabele Hoeveelheid water	Afhankelijke variabele Hoogte van de plant
	

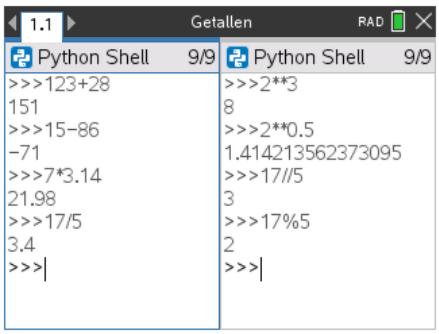
2. Getallen in Python

Python kent tal van verschillende getal-types. We beperken ons hier tot gehele, Integers (int), en decimale getallen, Floating Point (float).

2.1. Eenvoudig rekenwerk

De meest gebruikte operaties voor rekenwerk in Python zijn de onderstaande. We voeren enkele berekeningen uit in de Python Shell

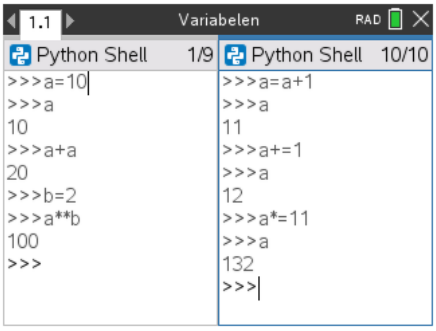
Operator	Bewerking
+	Optelling
-	Verschil
*	Vermenigvuldiging
/	Deling
**	Machtsverheffing
//	Vloerdeling
%	Modulo (rest)



Bij het invoeren van een bewerking in de Shell of het gebruik in code wordt de operator rood weergegeven.

2.2. Declaratie van variabelen

In Python wijzen we een waarde (data) toe aan een variabele met het gelijkheidsteken, =.

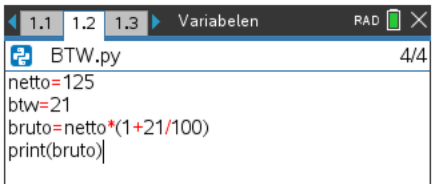


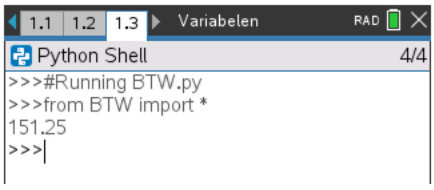
Bij een herdeclaratie van de variabele a kan de uitdrukking a = a + 1 verkort noteren met a += 1. Hetzelfde geldt voor -=, *=, /=, ...

Merk op dat na het ingeven van a = 10, de waarde niet getoond wordt.

Als we het volgende programma dat de nettoprijs (exclusief BTW) omzet in de brutoprijs (inclusief BTW) runnen, krijgen we het volgende resultaat in de Python Shell.

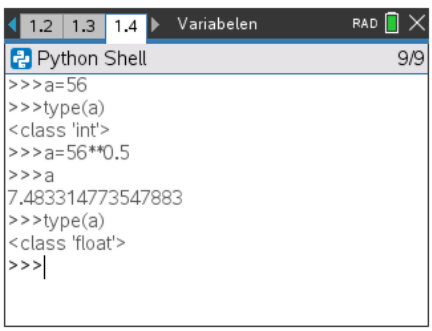
Om ook effectief de brutoprijs als output in de Shell te tonen, gebruiken we het print()-statement.





De naam van een variabele moet aan de volgende regels voldoen:

- niet starten met een getal
- geen spaties
- geen gebruik van de volgende symbolen:'",<>/?\()@#$%^&*~-+
- kleine letters is een algemene afspraak
- vermijd het gebruik van woorden met een speciale betekenis in Python: float, int, ...



Merk op dat Python dynamisch omgaat met datatypes voor variabelen.
M.a.w., je kan een variabele een herdeclaratie geven van een ander data type.

Het type van een variabele verifieer je met de type()-statement.

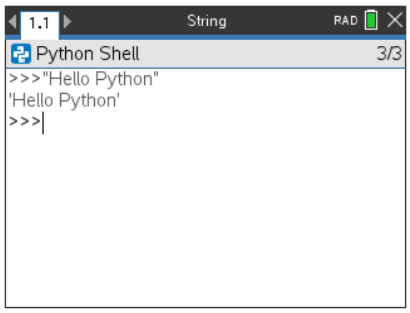
3. Woorden in Python

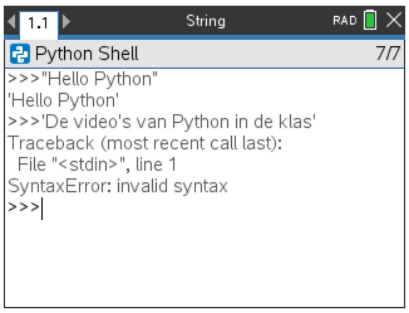
Woorden of Strings worden in Python gebruikt om tekst te bewaren. Een string is in feite niets anders dan een rij van karakters in een specifieke orde. Hetgeen betekent dat we m.b.v. indexering iedere letter van een woord kunnen aanspreken.

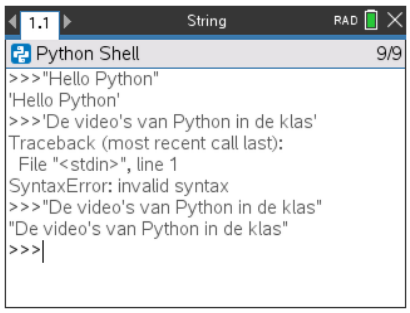
3.1. Definiëren van een string

Een string of woord plaats je tussen aanhalingstekens, b.v. “Hello Python”. Je kan ook gebruik maken van enkele aanhalingstekens maar dat geeft niet altijd het gewenste resultaat: ‘De video’s van Python in de klas’.

De error komt omdat het aanhalingsteken van video’s de string afsluit. Het gebruik van een enkel aanhalingsteken kan perfect binnen dubbele aanhalingstekens: “De video’s van Python in de klas”.



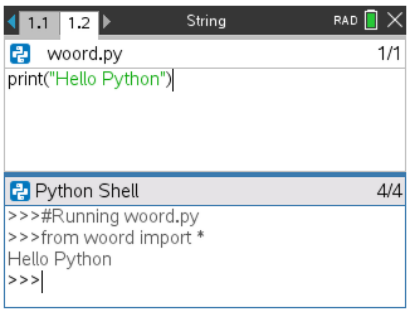




3.2. Printen van een string

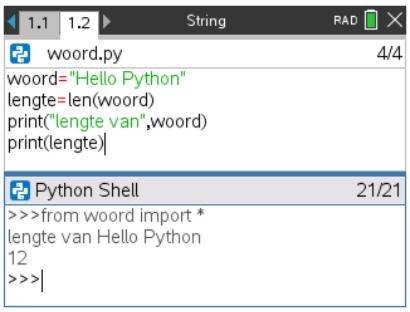
Het print()-statement kan ook hier gebruikt worden voor het tonen van een string van uit een programma.

Onderstaande print()-statements kunnen samengevoegd worden door gebruik te maken met de \n-operator die staat voor een nieuwe lijn.



3.3. Lengte van een string

Met de functie len() kan je het aantal karakters van een string controleren.

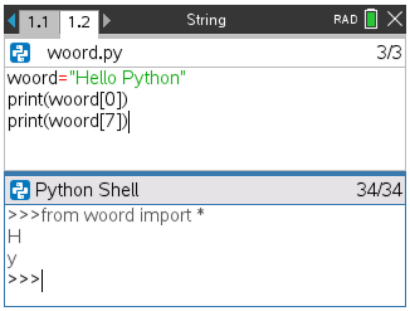


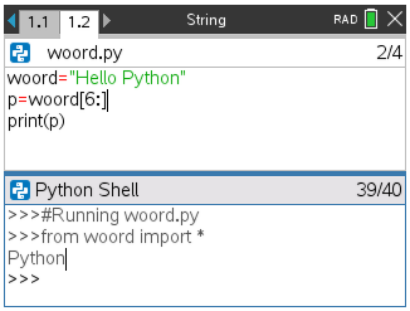
3.4. Indexering

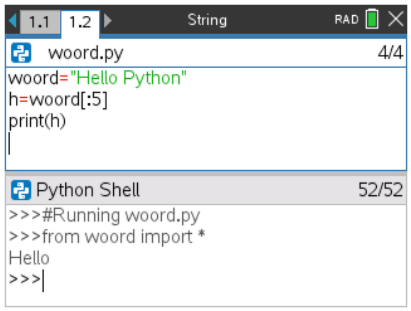
Zoals eerder aangeven is de volgorde van de karakters in een woord vast. Ieder karakter kan aangesproken worden door de index die de positie weergeeft van het karakter in de string.

Python gebruikt vierkante haakjes voor het aangeven van een index. Merk op dat het eerste karakter van een woord overeenkomt met index 0.

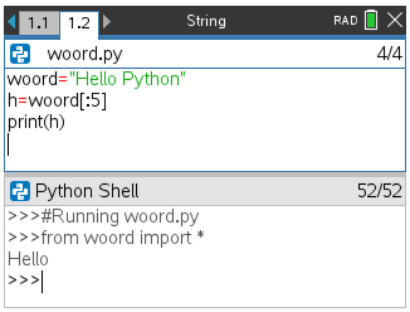
Men kan ook vanaf of tot een bepaalde index alles weergeven en/of bewaren in een andere variabele. In Python noemt men dit slicing. In Python betekent woord[:3], neem alle karakters (elementen) van index 0 tot 3, index 3 niet inbegrepen.

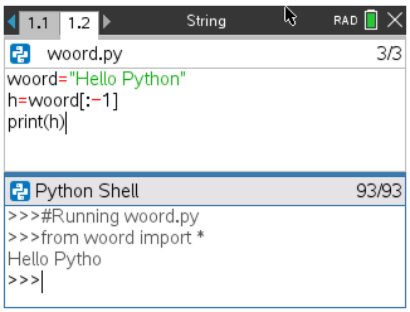


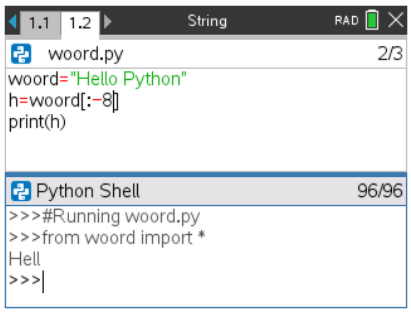




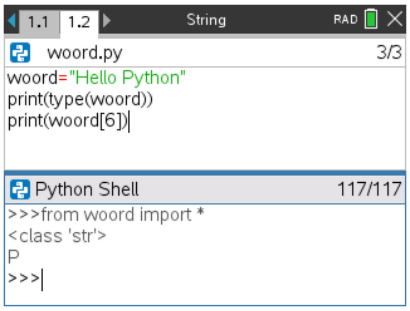
Met negatieve slicing kunnen we tellen vanaf het einde van een woord.

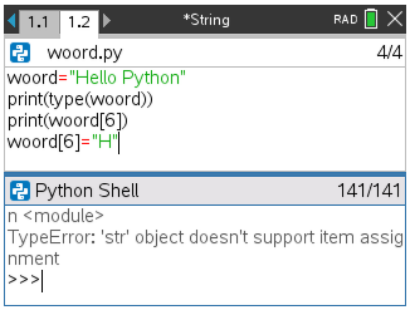


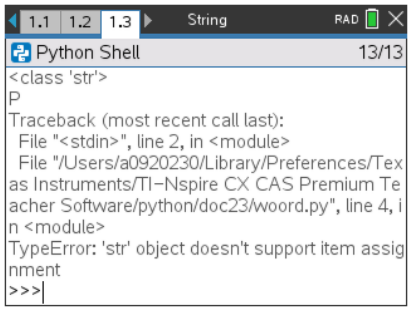




Strings in Python hebben de eigenschap onveranderlijkheid: als een variable gedeclareerd is als een string, kunnen de karakters van de varaiable (string) niet worden veranderd.







3.5. String-methodes

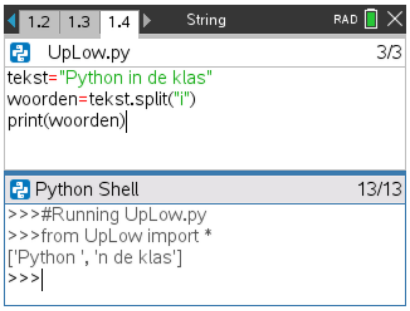
In Python is alles een object: een specifiek getal is een object van b.v. de klasse int (geheel getal) of float (decimaal getal) en “Hello Python” een object uit de klasse str (string). Wat dit exact betekent, verduidelijkingen we meer in details in het gedeelte Object georiënteerd programmeren.

Voor iedere klasse van objecten, zijn er ingebouwde methodes (of functies) die op de objecten kunnen uitgevoerd worden. Voor het uitvoeren van een methode op een object gebruiken we een punt gevolgd door de methodenaam: object.methode() of object.methode(parameters).

Enkele voorbeelden voor de string tekst = ”Python in de klas”.

upper()	alle karakters in hoofdletters
lower()	alle karakters in kleine letters
split()	splitsen bij een spatie
split(“i”)	splitsen bij een specifiek element (element niet inclusief)

“Python in de klas”.split() geeft als resultaat [‘Python’,’in’,’de’,’klas’], een lijst met alles element de woorden van de tekst. Hetgeen ons brengt tot het volgende data type Lijsten.

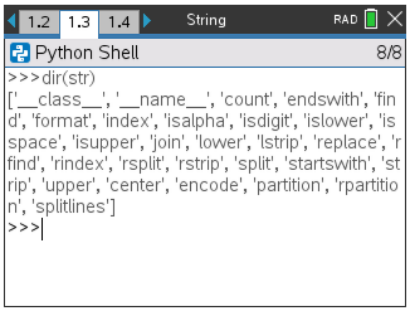


Voor een overzicht van alle ingebouwde methodes voor een bepaalde klasse, voer de dir()-statement uit in de Python Shell.

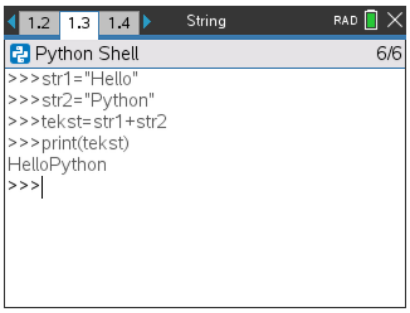
Wat en hoe over deze methodes kunt u vinden @ www.python.org of www.micropython.org.

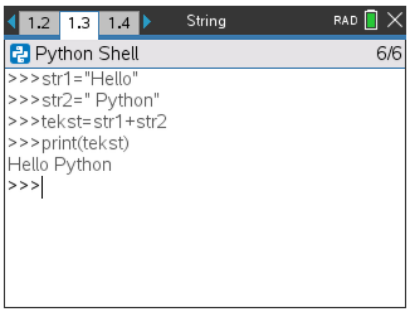
Of natuurlijk uitproberen is ook een zeer zinvolle leerschool.

De implementatie van Python in TI-technologie is gebaseerd op MicroPython, versie 3.4.



Het samenvoegen van twee strings kan heel eenvoudig door gebruik te maken van de +-operator.





4. Lijsten in Python

Lijsten kan je beschouwen als de meest algemene versie van een rij in Python. Zoals voor strings is de volgorde van belang maar lijsten zijn veranderlijk, m.a.w. ieder element van een lijst kan, na definitie, veranderd worden gebruikmakend van de index van een element.

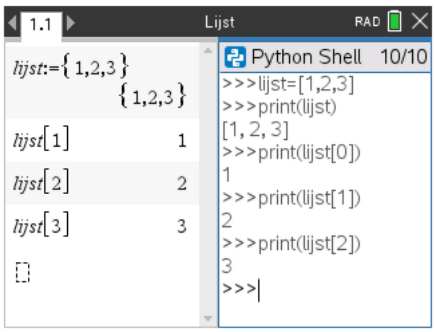
4.1. Definiëren van een lijst

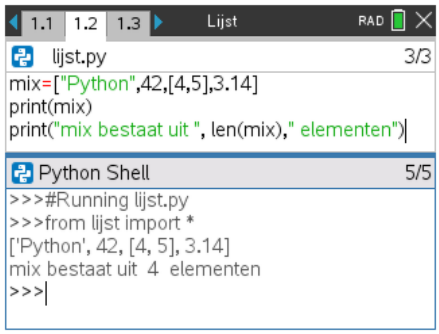
Het grote verschil met lijsten voor TI-technologie is dat de indexering van lijsten in Python start vanaf 0 i.p.v. vanaf 1 zoals voor TI-technologie.

Bovendien is de syntax van een lijst verschillend:

- Python lijst = [1,2,3,4,5]
- TI Apps lijst := {1,2,3,4,5}

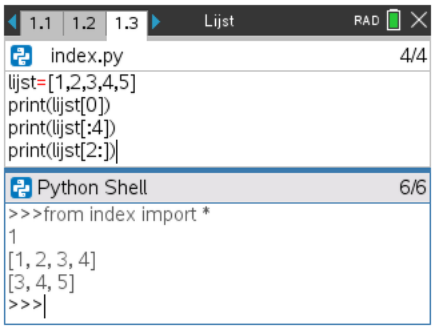
Python lijsten kunnen verschillende soorten van data types bevatten. Voor lijsten bepaalt het len()-statement ook het aantal elementen van een lijst.

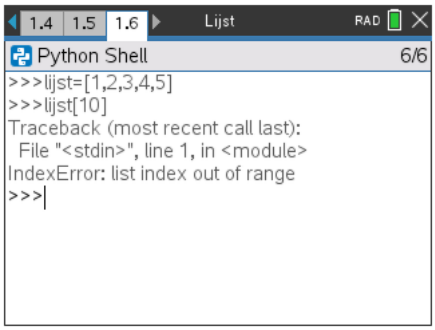




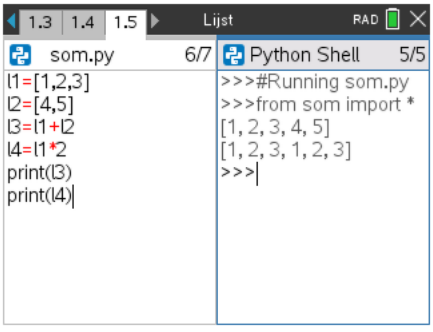
4.2. Indexering & Slicing

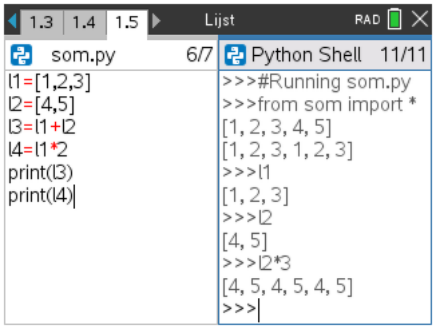
Indexering en slicing voor lijsten werkt hetzelfde als voor strings, hieronder enkele voorbeelden. Indien er geen element is voor een bepaalde index geeft Python de volgende error-boodschap.





Lijsten kunnen in Python eenvoudig samengevoegd worden m.b.v. +. En met * kun je een lijst verdubbelen.
Merk op dat deze operatie op lijsten de originele lijsten niet verandert, en om het resultaat te bewaren er een nieuwe variabele moet gedeclareerd worden.

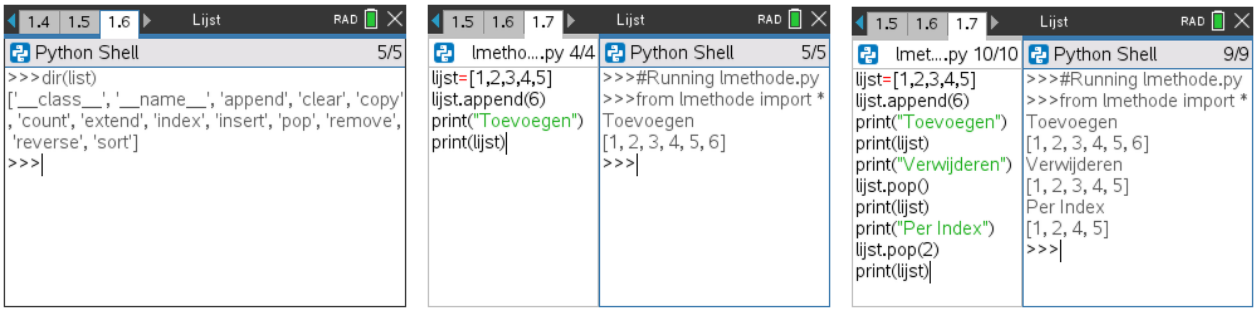




4.3. Lijst-Methodes

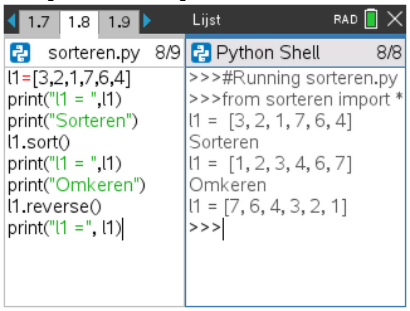
Hieronder enkele methodes die uitgevoerd kunnen worden op lijsten.

- append() voeg een element permanent toe aan het einde van de lijst
- pop() verwijder permanent het laatste element, of specifiek pop(1)
- pop(i) verwijder permanent het element met index i

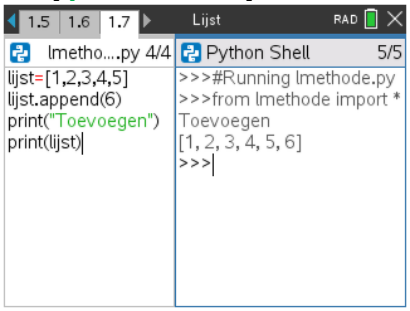


We bekijken ook even de methodes sort() en reverse(). Het manipuleren van een lijst verandert de lijst ook hier permanent.

l1=[3,2,1,7,6,4]

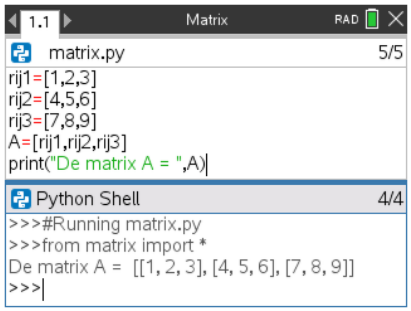


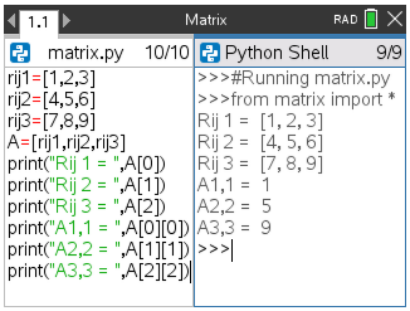
l1=["y","x","z","t","u"]



4.4. Matrices

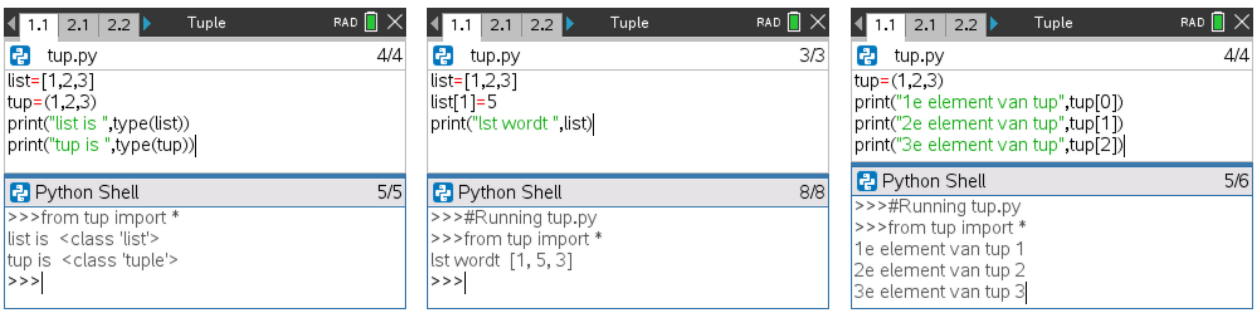
Matrices kunnen voorgesteld worden gebruikmakend van een lijst bestaande uit lijsten.



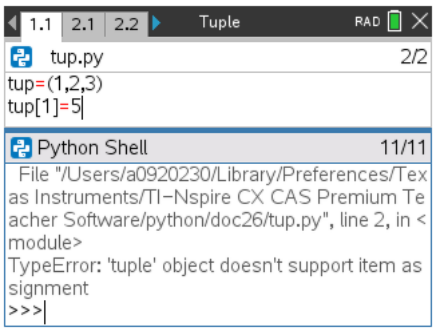


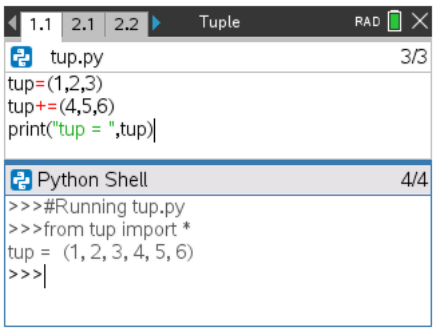
5. Tuples

Het data type Tuple is vrij gelijkaardig als een lijst alleen dat, zoals voor strings, de elementen van een tuple niet kunnen veranderd worden.

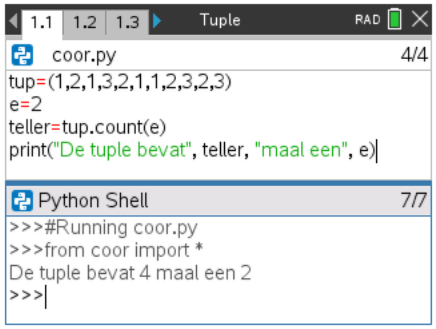


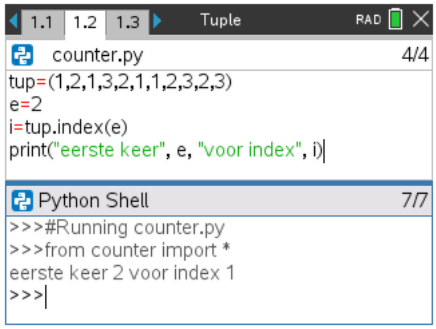
Indien we een element van een tuple proberen te veranderen, b.v. tup[1]=5, krijgen we de onderstaande error-boodschap. Wel kunnen elementen toegevoegd worden met de + operator.





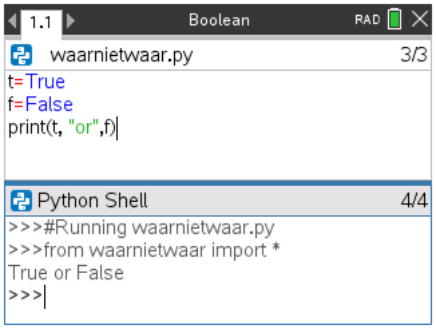
Voor tuples hebben we slechts twee methodes beschikbaar: count en index.

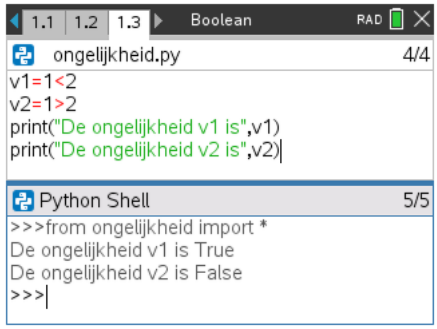




6. Boolean - True or False

Python heeft twee voorgedefinieerde booleaanse objecten: True en False. True en False zijn in feite niet anders dan de getallen 1 en 0.

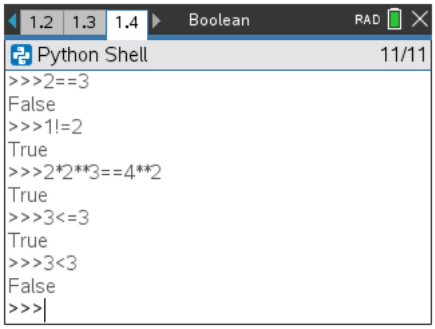




6.1. Vergelijkingsoperatoren

Hieronder de lijst met vergelijkingsoperatoren.

Operator	Bewerking
==	gelijk aan
!=	niet gelijk aan
>	groter dan
<	kleiner dan
>=	groter of gelijk
<=	kleiner of gelijk

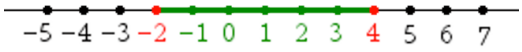


6.2. And en/of Or

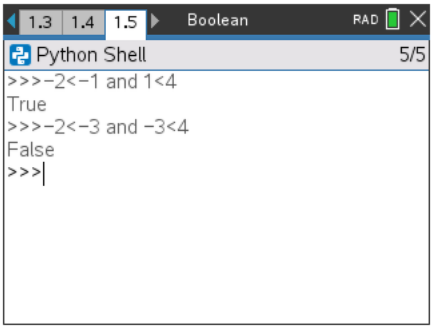
Met de statements and en or kunnen verschillende logische tests met mekaar gecombineerd worden om meer complexe testen te construeren.

A	B	A and B	A or B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

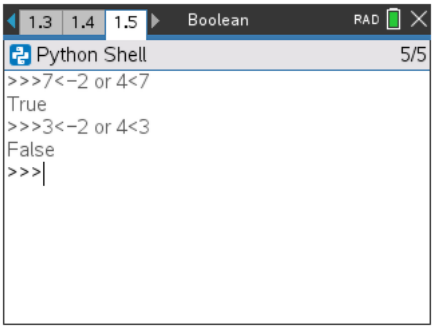
Enkele voorbeelden



$|x-1|<3 \Leftrightarrow -2<x$ en $x<4$



$|x-1|>3 \Leftrightarrow x<-2$ of $4<x$



7. Input

Met het input()-statement kan via de shell input geleverd worden aan een Python program, b.v. als volgt:

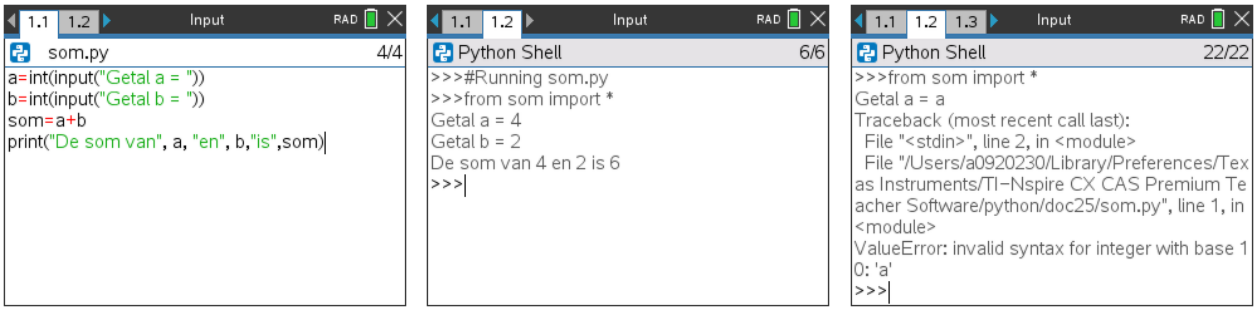
Hiernaast hoe stap voor stap

- Input van de twee getallen
- Berekenen van de som
- Tonen van de som

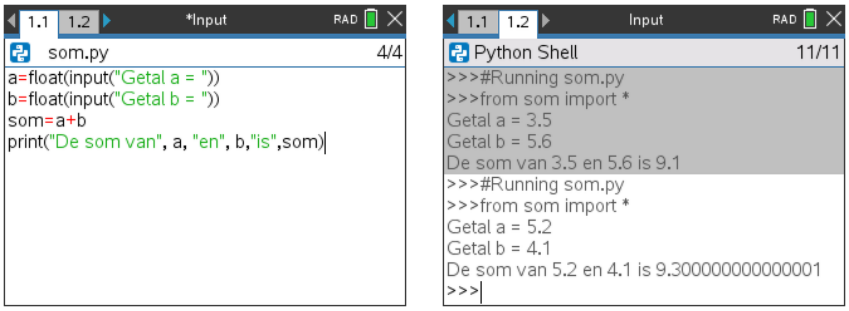
Alleen is de uitkomst van de som 4 + 2 niet echt wat we verwachten van een rekenkundige som.

Input in Python wordt altijd geïnterpreteerd als strings.

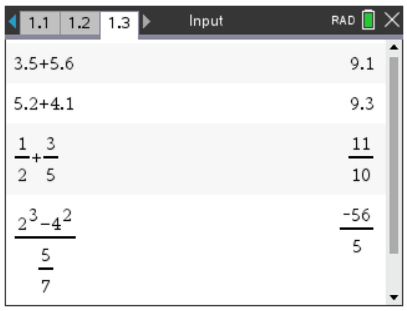
Om dit op te lossen, maken we gebruik van de ingebouwde Python-functie int(). Als we bij de input niet een geheel getal ingeven, krijgen we de onderstaande foutmelding. Hoe we een foutmelding bij een verkeerde input kunnen voorkomen, bekijken we later in deze bootcamp.



Indien we willen werken met decimale getallen vervangen we int() door float().



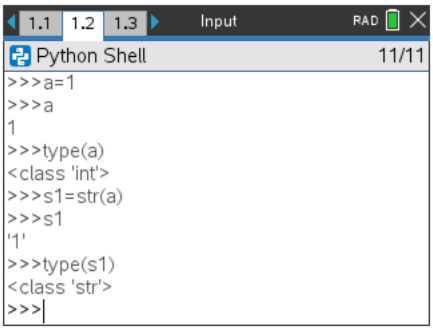
Wiskundig rekenen met decimale getallen is niet de sterkste kant van de programmeertaal Python. Het resultaat van een tweede run van het programma som.py geeft het resultaat 9.300000000000001.

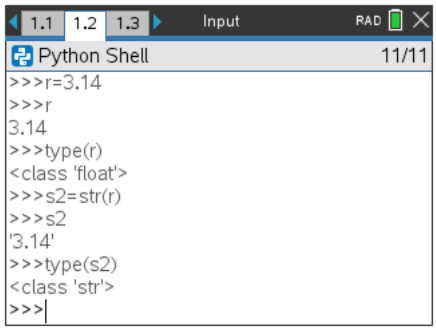


Dit heeft te maken met het feit dat Python gebruik maakt van binaire benaderingen afhankelijk van de hardware. Meer info hierover: docs.python.org/3/tutorial/floatingpoint.html.

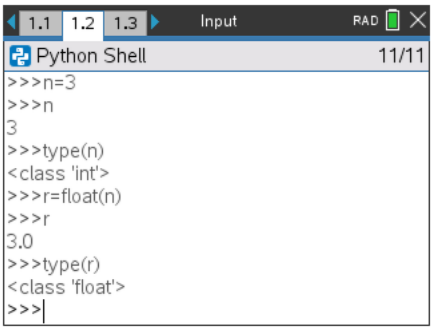
TI-technologie beschikt over een sterke wiskundige engine (numeriek of CAS) die best gebruikt wordt voor het uitvoeren van wiskundige berekeningen.

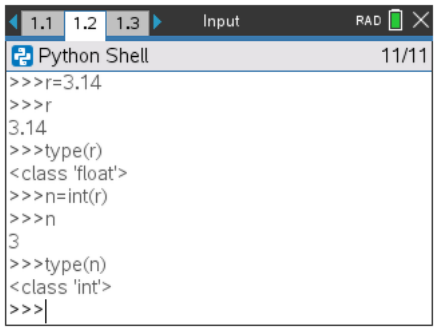
Omgekeerd kunnen getallen ook omgezet worden in strings met de str()-statement.





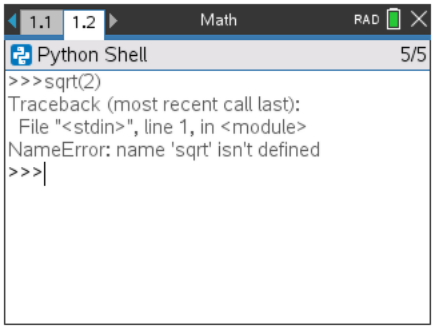
int() en float() kunnen ook gebruikt worden om gehele getallen om te zetten in een decimaal getallen en omgekeerd.





8. Extra wiskundige functionaliteit

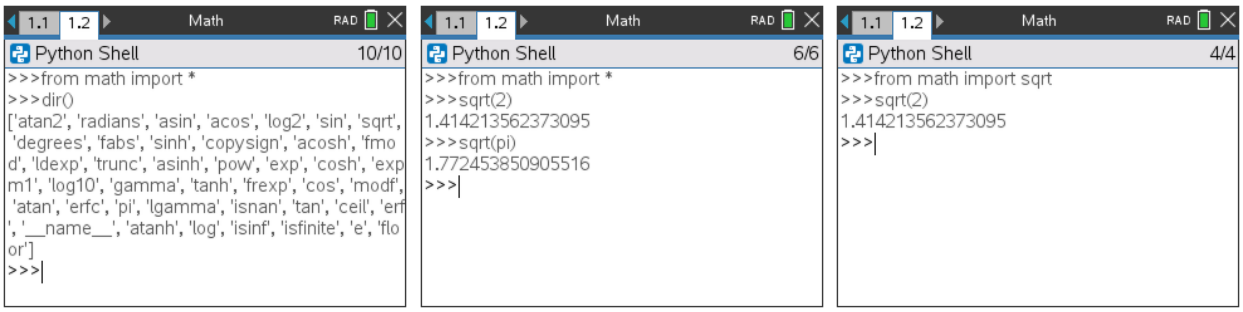
Indien we sqrt(2) willen berekenen, krijgen we de onderstaande foutmelding, m.a.w. de functionaliteit vierkantswortel in geen ingebouwde functie.



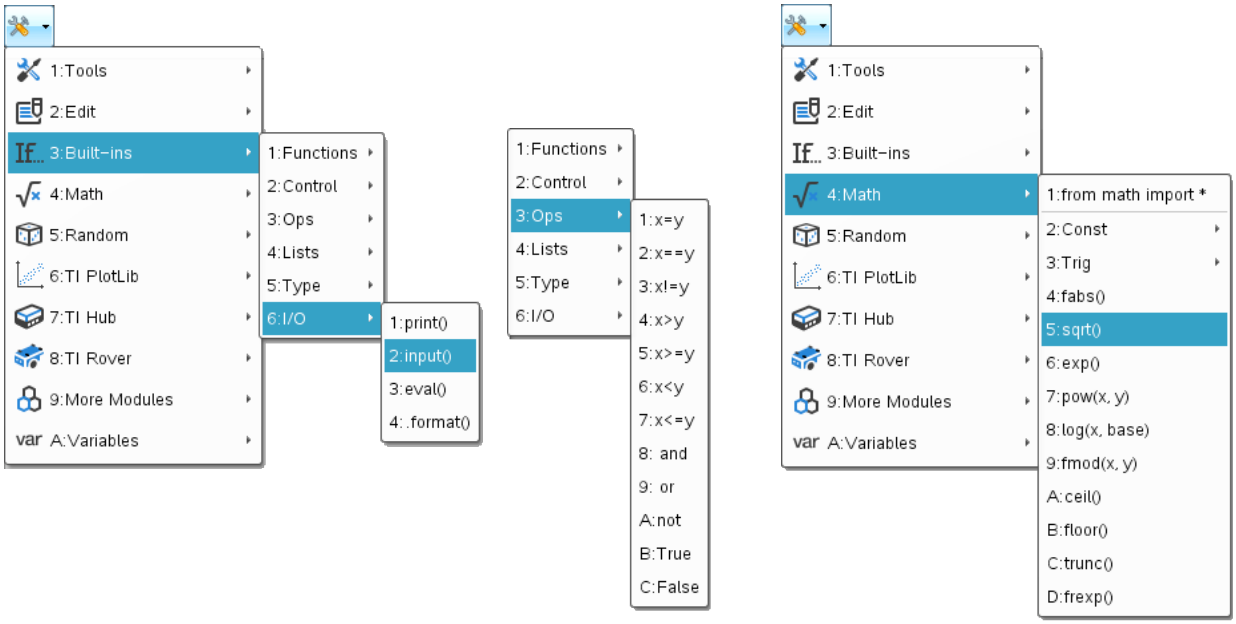
8.1. De module Math

Na het importeren van de module Math, met de code “from math import *”, kunnen we gebruik maken van heel wat extra wiskundige functies. dir() produceert een lijst met alle functionaliteit in de module Math.

Indien je b.v. enkel sqrt() uit de module Math wil importeren, gebruik je de code “from math import sqrt”.



De meeste gebruikte functie, zowel voor de built-in functies als voor de geïntegreerde modules, zijn beschikbaar in een menustructuur.

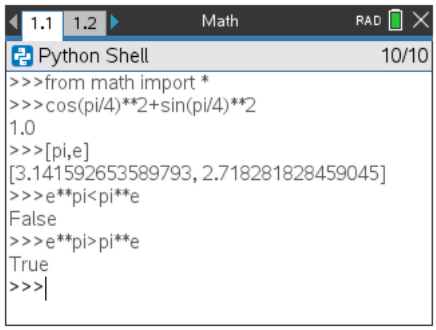


Door deze menustructuur is het eenvoudig om commando’s en functies terug te vinden, versnelt het ingeven van code (zeker voor het programmeren met de handheld) en gebruikt je steeds de juiste syntax.

Nog enkele andere voorbeelden van de module Math.

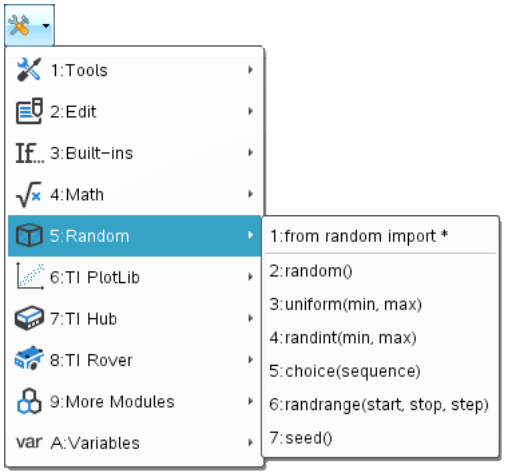
e^{π} or π^{e}

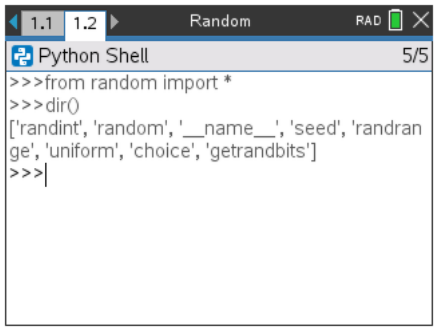
Which One is Greater?



8.2. De module Random

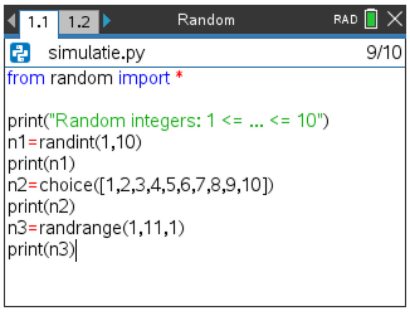
De module Random bevat functies die gebruikt kunnen worden om simulaties uit te voeren met een Python programma.

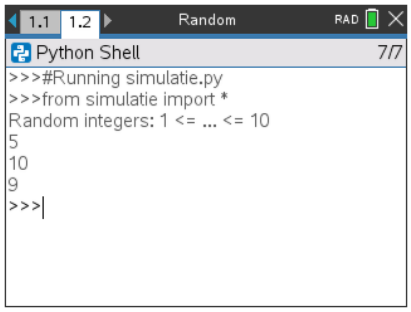


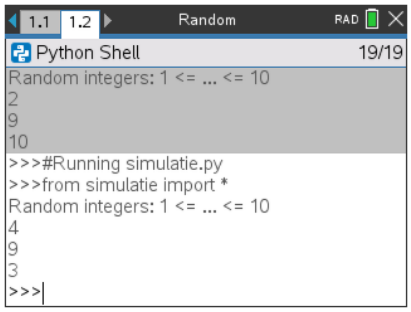


De onderstaande code genereert telkens een willekeurig geheel getal n tussen 1 en 10, 1 en 10 inclusief.

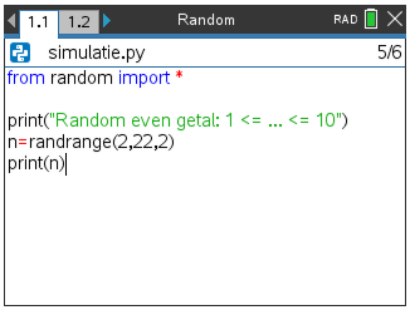
- randint(min,max) random geheel getal tussen min en max: min <= … <= max
- choice(lijst) random keuze uit een lijst
- randrange(start,stop,step) random geheel getal tussen start en stop met stapgrootte step

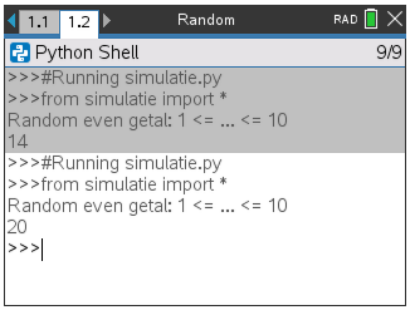


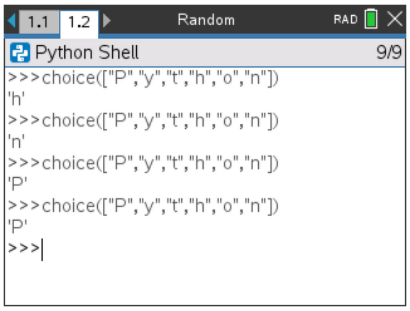




Met randrange kunnen we at random een even getal generen tussen tussen 1 en 20, 20 inbegrepen: randrange(2,22,2). Merk op dat de choice()-statement ook andere data kan bevatten dan gehele getallen.







9. Print

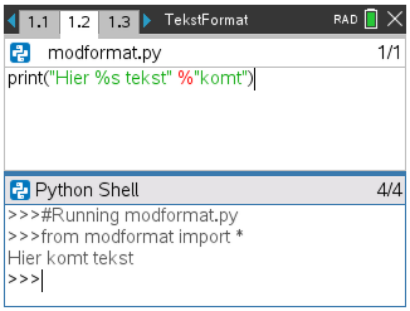
We hebben het print()-statement reeds gebruikt om de waarde van een variabele weer te geven, al dan niet in combinatie met tekst.

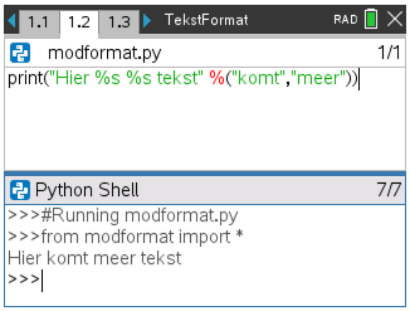
We bekijken twee string-methodes om de output van een Python programma te formateren:

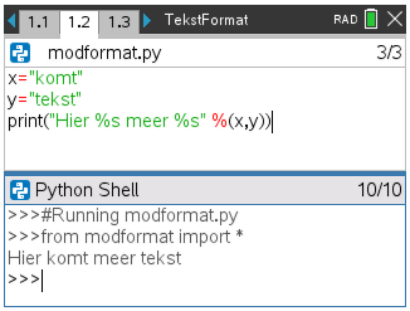
- % operator (de oudste)
- format()-methode (de verbeterde)

9.1. Formateren met de %-operator

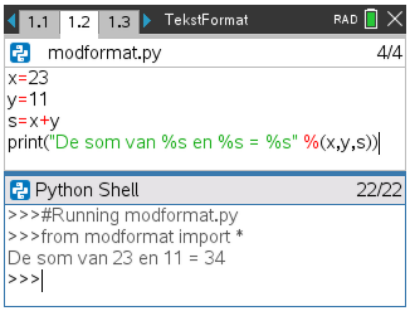
Enkele voorbeelden



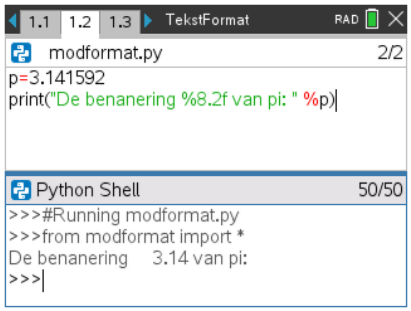




Tuurlijk geldt dit niet alleen voor tekst, maar ook voor andere data types zoals getallen.

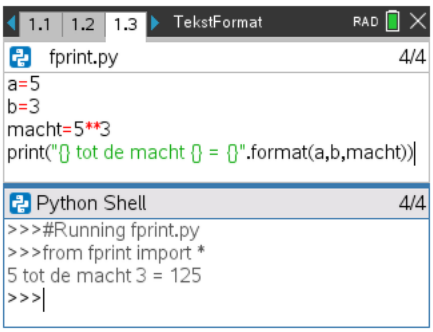


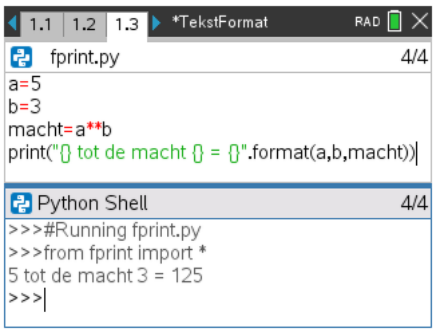
De format %8.2f betekent dat we minimum 8 karakters gebruiken om het getal weer te geven (eventueel opgevuld met spaties) en .2f bepaalt dat er slecht twee cijfers na de komma worden weergeven.



9.2. Formateren met format()

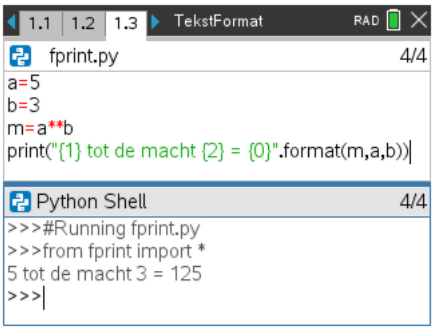
Een betere manier om strings te formateren in een print()-statement is gebruik te maken van de format()-methode: print("{} tot de macht {} = {}".format(a,b,macht)).



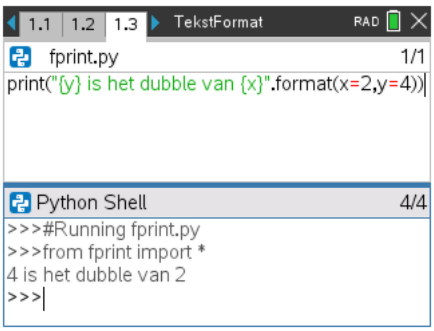


Voordelen van het gebruik van de format()-methode zijn o.a.:

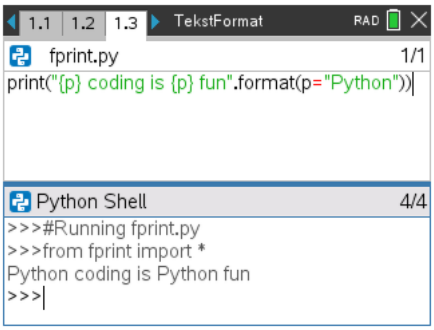
- In te voegen objecten kunnen geïndexeerd worden



- In te voegen objecten kunnen geassocieerd worden met labels

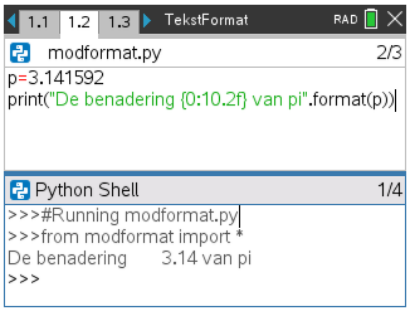


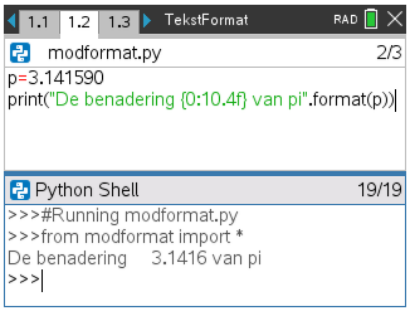
- In te voegen objecten kunnen meervoudig gebruikt worden

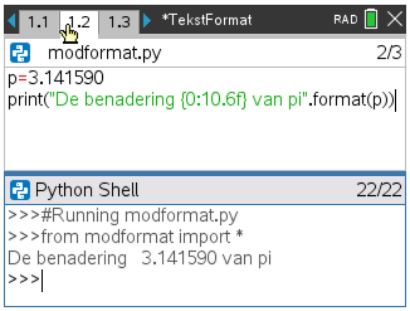


Het bepalen van het aantal decimalen na de komma met de format()-methode gaat als volgt.

Voor {0:10:2f} is 0 de index van het object,
10 het aantal minimale karakters (eventueel opgevuld met spatie) en
2f het aantal decimalen na de komma.

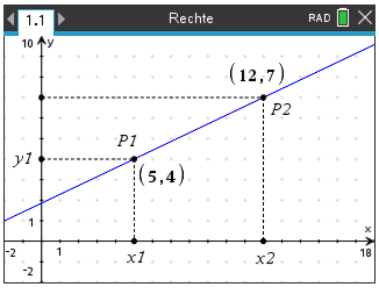






10. Inleidend voorbeeld

We tekenen een rechte in de TI-Nspire CX Graphs App door de punten $P1$ en $P2$.



We bewaren de coördinaten van de punten $P1$ en $P2$ in de volgende TI-Nspire CX variabelen $x1, y1$ en $x2, y2$ als volgt:

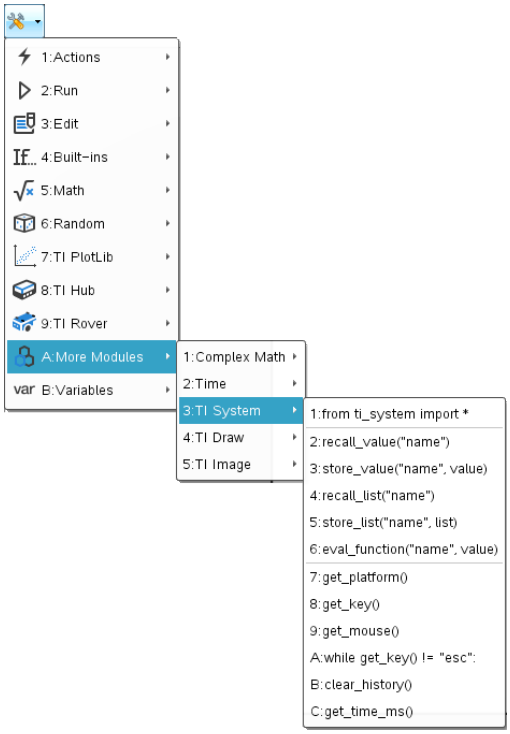
$P1\ (x1, y1)$ & $P2\ (x2, y2)$.

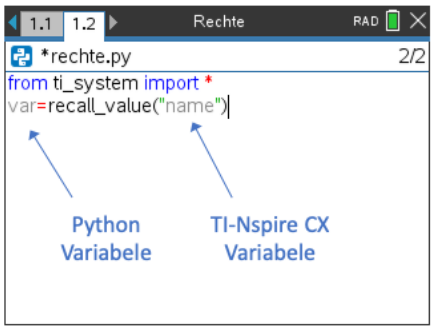
De vergelijking van de rechte is gelijk aan:

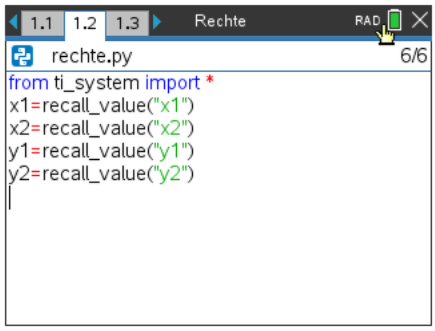
$$y - y1 = \frac{y2-y1}{x2-x1}(x - x1).$$

Met de module TI System kan variabelen vanuit TI-Nspire CX geïmporteerd worden in een Python-programma als Python variabelen. Na het importeren van de module ti_system selecteren we het commando recall_value().

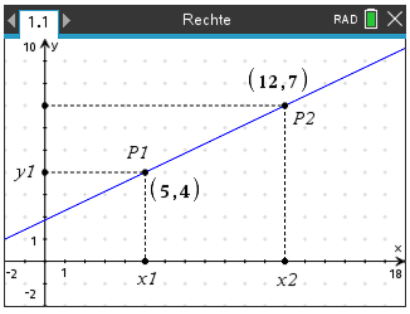
De onderstaande syntax-template verschijnt die aangeeft hoe de Python variabele te declareren op basis van de TI-Nspire CX variabele.

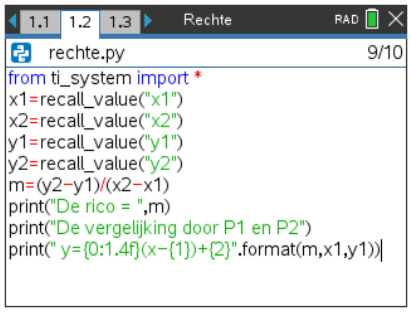




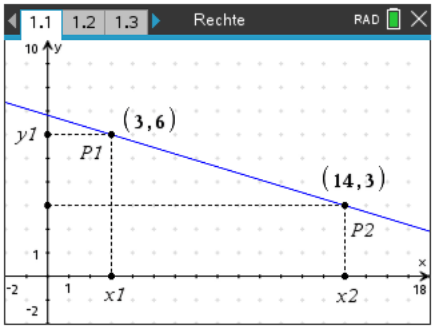


We berekenen de richtingscoëfficiënt m en bepalen de rechte door de twee punten $P1$ en $P2$.

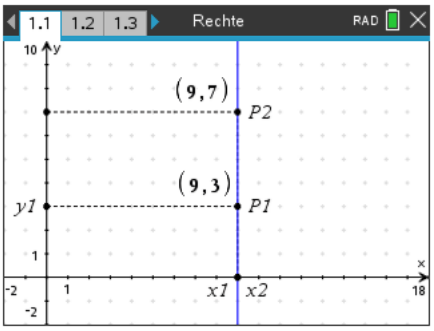


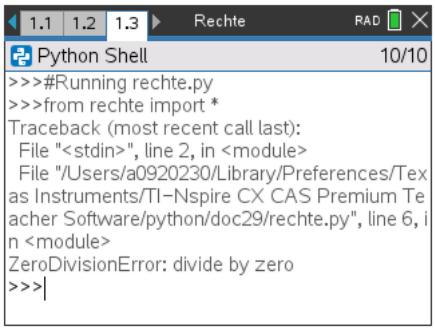


Als we de punten verplaatsen en het programma opnieuw runnen krijgen we de nieuwe vergelijking:



Maar wat met een verticale rechte?





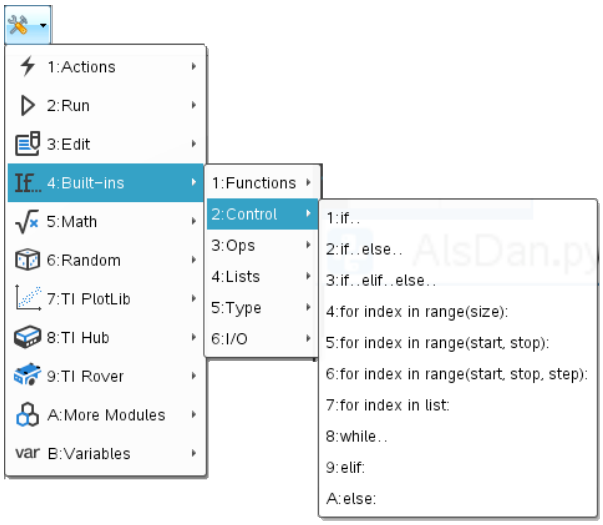
11. If-Statements

Om bovenstaande foutmelding te voorkomen testen we de conditie x1 == x2 of x1 != x2.

Hiervoor selecteren we in het Control-menu de optie 2: if..else.. en passen ons programma als volgt aan.

Om commentaar in te voeren start je de regel met #.

De template van het if-statement verschijnt automatisch in de Program Editor.

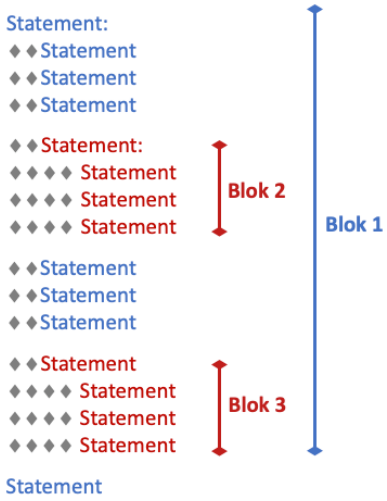


```
if BooleanExpr:
    block
else:
    block
```

Een if-statement bestaat uit het sleutelwoord if, gevolgd door een voorwaarde en een dubbelpunt. De statements na het dubbelpunt, moeten in een blok staan. Als de voorwaarde True is, wordt alle statements in het blok uitgevoerd.

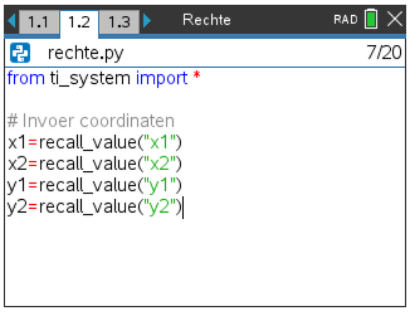
Een blok is een verzameling van gegroepeerde statements. De Whitespace voor de statements geef aan welke statements tot eenzelfde blok. Statements die op dezelfde positie starten, m.a.w. regels met dezelfde inspringingen, behoren tot hetzelfde blok.

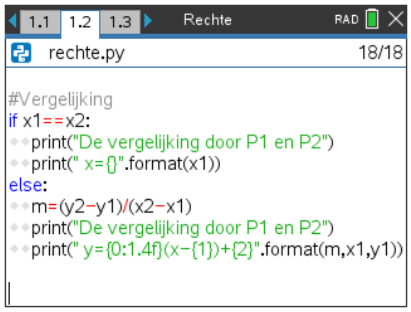
Merk op dat if-statements niet afgesloten worden met een End-statement maar begin en einde is gebaseerd op gelijke inspringingen. In TI-Technologie wordt de Whitespace aangeduid met ♦♦.



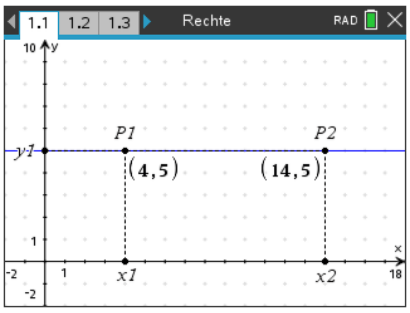
Een if-statement kan uitgebreid worden met een else-statement met eventueel een elif-statement tussenin.

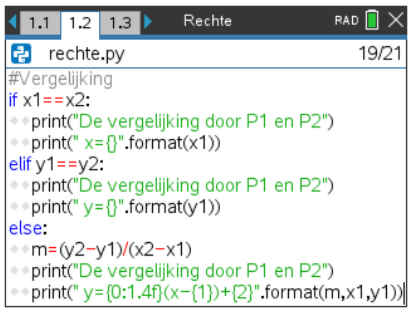
Voor ons programma geeft dit het volgende:





Met een extra elif-statement kunnen we ook de vergelijking van een horizontale rechte eleganter weergeven:

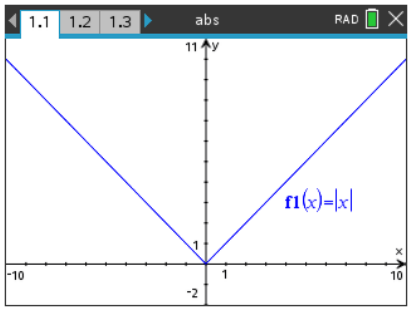


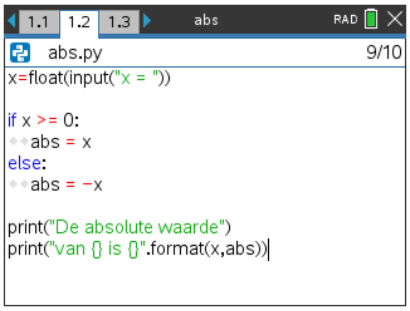


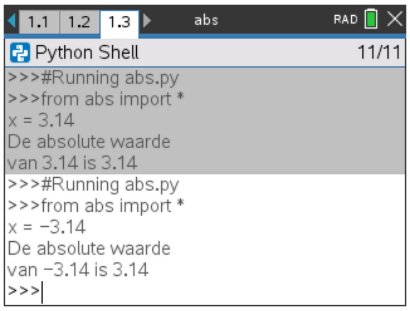
Om de structuur van Python-code goed op een rij te hebben, is het belangrijk steeds aandacht te hebben op hoe de structuur van een Python-programma gebaseerd is op inspringingen (indents of whitespace).

Nog twee voorbeelden om de structuur van de if-statements te illustreren.

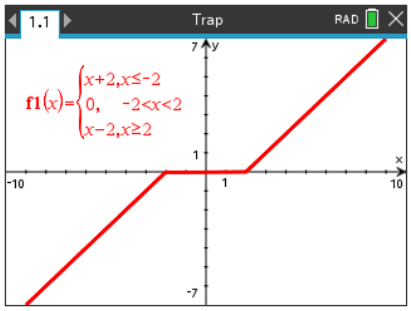
1. Bepaal de absolute waarde van een reëel getal baserend op de definitie $|x| = \begin{cases} x \ als\ x \leq 0 \\ -x \ als\ x < 0 \end{cases}$.

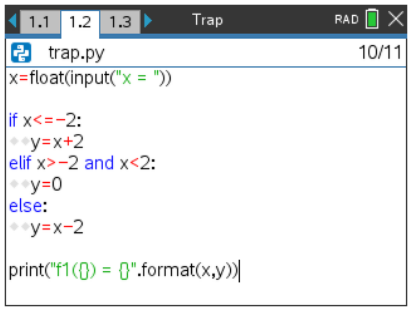


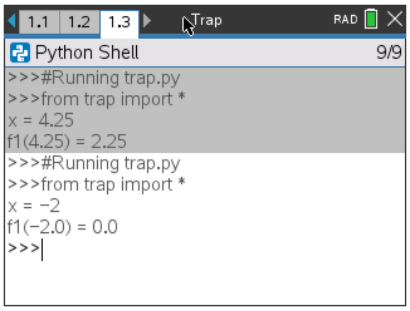




2. Bereken de functiewaarde voor de volgende stuksgewijs gedefinieerde functie $f1(x) = \begin{cases} x+2 \ als\ x \leq -2 \\ 0 \ als -2 < x < 2 \\ x-2 \ als\ x \geq 2 \end{cases}$.





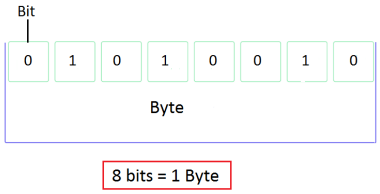


Programmeeropdrachten

Opdracht 1
Schrijf een programma dat als output het gehele quotiënt en de rest geeft van twee gehele getallen a en b.

Opdracht 2
Schrijf een programma dat als output het maximum van twee reële getallen r en s geeft.

Opdracht 3: Binaire getallen



Bit
Ook gekend als Binary Digit. Het is de kleinste eenheid van informatie die bewaard Kan worden op of gemanipuleerd door een computer. Een bit is ofwel 0 of 1.

Byte
Een byte is een groep van 8 bits. Oudere computers konden enkel 8 bits tegelijk bewerken. Iedere Byte kan 1 karakter – 'A', 'k', '@', … – bewaren. $2^8 = 256$, van daar dat de uitgebreide ASCII-code tabel 256 (0-255) karakters bevat.

In Python worden binaire getallen voorgesteld met het voorvoegsel 0b gevolgd door een binair getal:

```
>>>bin=0b00001011
>>>bin
11
```

Python kent de bin()-functie om een decimaal getal om te zetten in een binair getal.

a. Ga na dat de uitvoer van deze functie van het type string is.

b. Schrijf een programma dat om een decimaal getal tussen 0 en 255 vraagt en als uitput een string geeft in de vorm zonder het voorvoegsel "0b": b.v. 27 wordt "11011".

Een 8-bits getal bestaat uit 8 nullen en/of enen.

c. Pas je programma aan zodat de uitvoer een string is die alle 8 bits geeft.
Het getal 27 wordt dan "00011011".

Opdracht 4: Schaar – Papier – Steen
Bij het spelletje *Schaar-papier-steen* kiezen twee spelers tegelijkertijd een van de drie objecten.

Kiezen beide hetzelfde, dan is de uitslag onbeslist.

In het andere geval wint schaar van papier, papier van steen en steen van schaar.

Schrijf een programma dat een speler tegen de computer laat spelen.

a. Begin met een input-opdracht in de vorm van:
```
k=int(input("0=schaar, 1=papier, 2=steen: "))
```

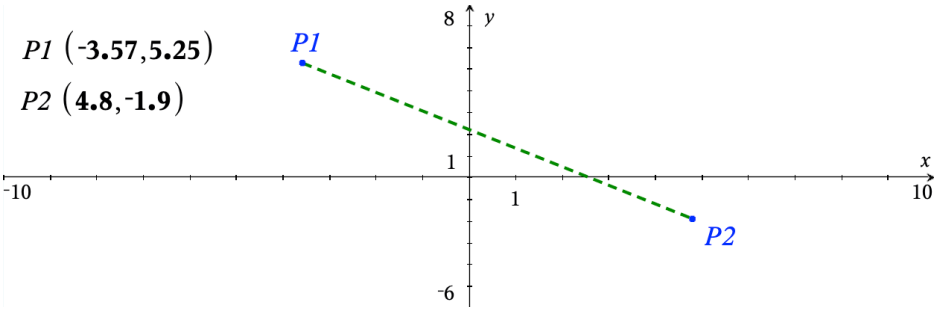
b. Laat vervolgens de computer een willekeurig getal kiezen tussen 0, 1 en 2.

c. Bepaal en druk de uitslag af.

Opdracht 5

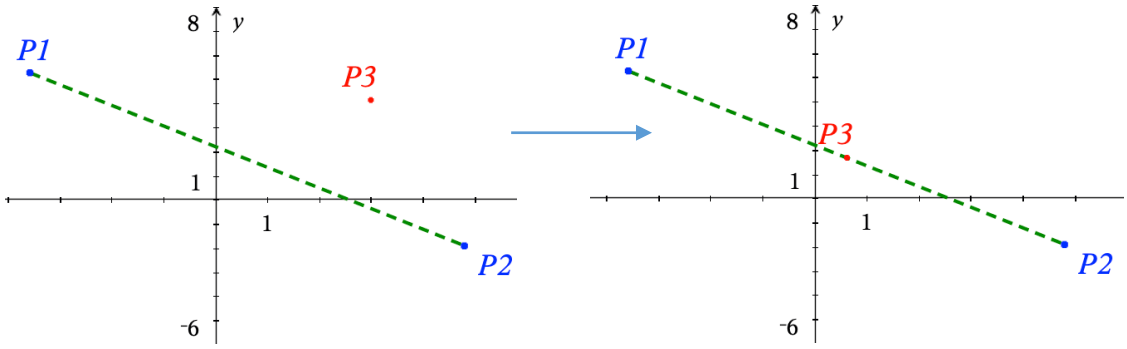
Teken twee punten $P1$ en $P2$ in de TI-Nspire CX graphs app (of gebruik het bijhorende tns-document).

Bewaar de coördinaten van de punten als volgt: $P1\ (x1, y1)$ en $P2\ (x2, y2)$.

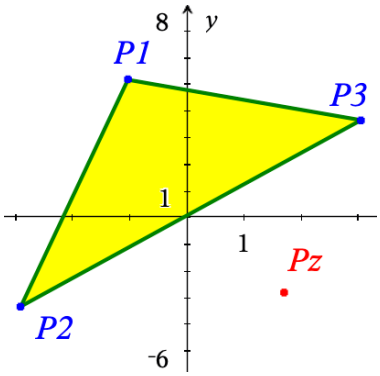


a. Schrijf een programma dat de afstand tussen de punten $P1$ en $P2$ (import coördinaten) berekent en als output de afstand geeft; tot op twee decimalen na de komma.

b. Schrijf een programma dat het midden $P3$ bepaalt tussen de punten $P1$ & $P2$ en $P3$ tekent in Graphs.
   - Teken een punt $P3$ in Graphs en bewaar de coördinaten als $P3\ (x3, y3)$.
   - Run dan het programma dat het midden berekent.



c. Schrijf een programma dat het zwaartepunt van een driehoek berekent en tekent.
Gebruik o.a. store_value("NSvar",pyVar) van de TI Sytem module.



Verdieping

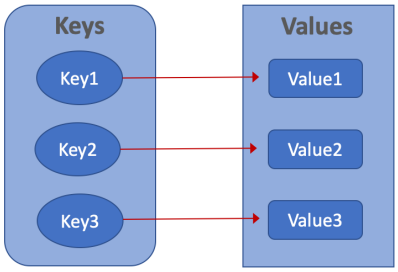
a. Dictionaries of woordenboeken

Een dictionary (data type: dict) is een verzameling van objecten die bewaard worden a.h.v. een sleutel (key). Iedere item van een dictionary bestaat uit een unieke key en een bijhorende waarde; waarbij deze waarde praktisch ieder Python object kan zijn.

Een belangrijk verschil met lijsten is dat voor dictionaries de volgorde van de items van geen belang is.

i. Syntax van een dictionary

Daar we elke key verbinden met een waarde creëren we een mapping-structuur bij het definiëren van een dictionary.

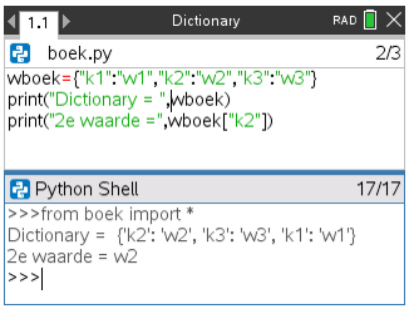


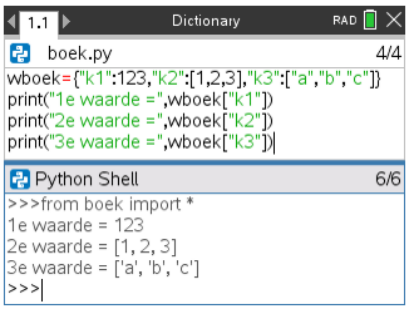
De syntax van een dictionary ziet er als volgt uit:

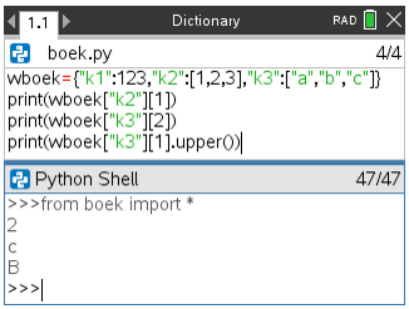
- De items van een dictionary staan tussen accolades { }
- Key en waarde worden gescheiden door een dubbele punt :
- De key staat altijd tussen aanhalingstekens “ “ (of ‘ ‘)

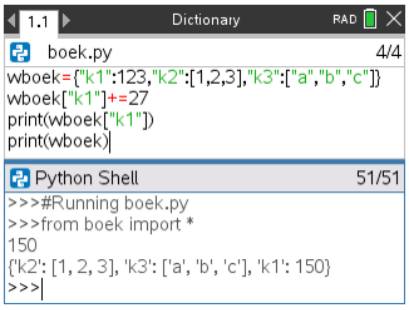
wboek={“key1”:”value1”, “key2”:”value2”, “key3”:”value3”}

We gebruiken de keys van een dictionary om een waarde aan te roepen, b.v. wboek[“key2”].



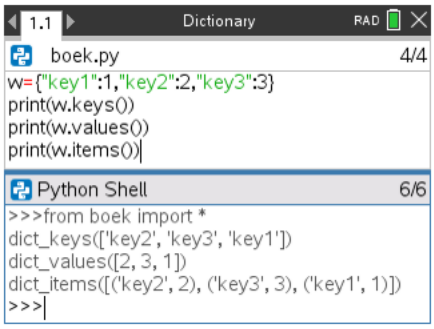


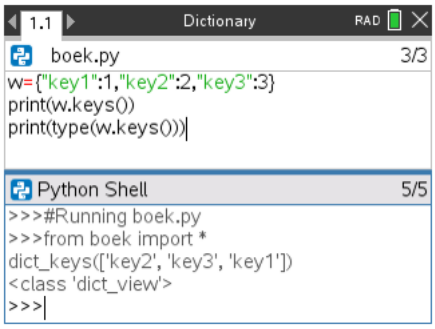




ii. Enkele methodes voor dictionaries

Hieronder de methodes om de lijst met keys, de lijst met waarden en de lijst met items te bekijken.



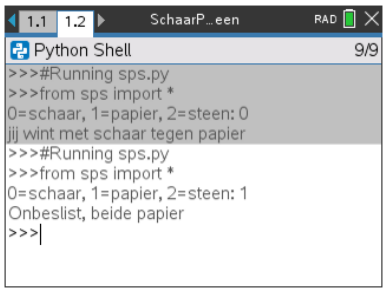


iii. Programmeeropdrachten

Opdracht 1: Schaar-Papier-Steen met dictionaries

Bij het spelletje *Schaar-papier-steen* is de uitslag onbeslist als beide spelers dezelfde keuze maken. In het andere geval wint schaar van papier, papier van steen en steen van schaar.

Hieronder een programma dat het spelen tegen de computer simuleert:

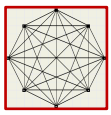



```
from random import *

keuzes=["schaar","papier","steen"]

k=int(input("1=schaar, 2=papier, 3=steen: "))
speler=k-1
comp=randint(0,2)

if speler==comp:
    print("Onbeslist, beide "+keuzes[speler])
else:
    if speler==0 and comp==1:
        print("jij wint met",keuzes[speler],"tegen",keuzes[comp])
    elif speler==1 and comp==2:
        print("jij wint met",keuzes[speler]+"tegen",keuzes[comp])
    elif speler==2 and comp==0:
        print("jij wint met",keuzes[speler],"tegen",keuzes[comp])
    else:
        print("jij verliest met",keuzes[speler],"tegen",keuzes[comp])
```

Pas bovenstaand programma aan om gebruikmakend van een dictionary met de spelregels voor winst, de voorwaardelijke statements wat eleganter te coderen, b.v.

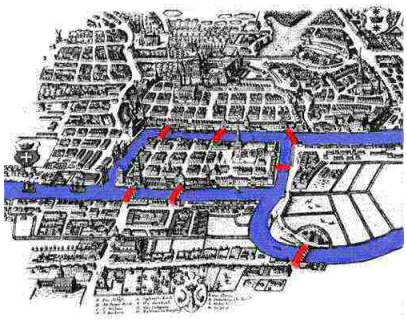
```
from random import *

keuzes=["schaar","papier","steen"]
regels={"schaar":"papier","papier":"steen",”steen":"schaar"}
……
```

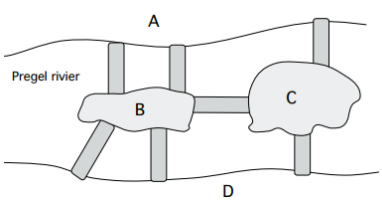
Opdracht 2: Grafen

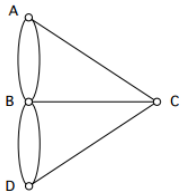


Een graaf is een verzameling van punten, knopen, waarvan sommige knopen verbonden zijn met lijnen, de zijden van de graaf. Grafen worden o.a. in de informatica gebruikt om het dataverkeer over netwerken weer te geven en te analyseren.

Eén van de eerste grafen-problemen is het probleem van de zeven bruggen van Koningsbergen: *Is het mogelijk een wandeling te organiseren zodat je precies één keer over iedere brug wandelt*. In 1736 loste Leonard Euler dit probleem op.

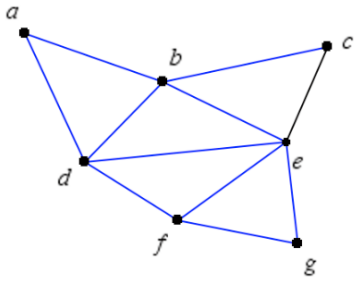






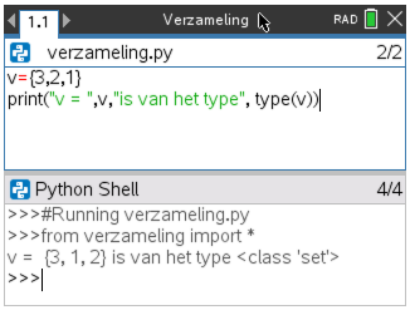
Euler bewees dat zo'n wandeling, een Euler pad, alleen mogelijk is als de graaf geen of exact twee oneven knopen heeft. Een knoop is oneven als er een oneven aantal zijden samenkomen.

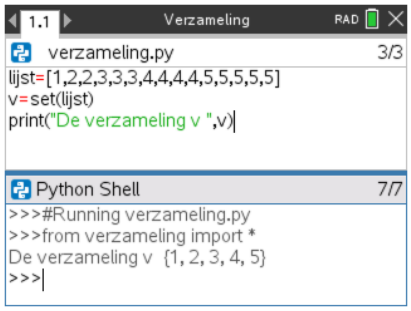
Schrijf een programma dat, gebruikmakend van een dictionary die als keys de knooppunten van de hiernaast afgebeelde graaf heeft, bij input van een knoop x als output een lijst van knopen geeft waarmee x verbonden is d.m.v. een zijde.

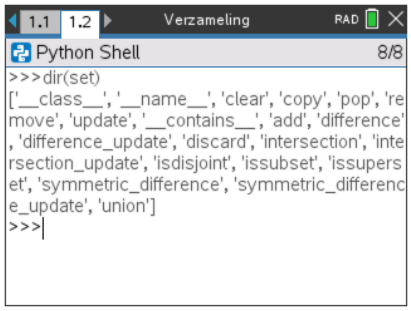


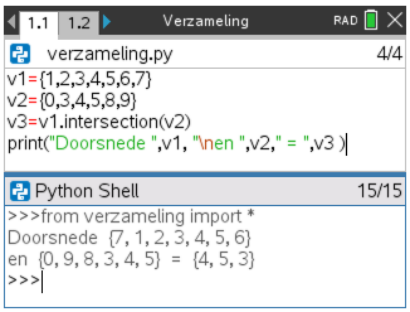
b. Sets of verzamelingen

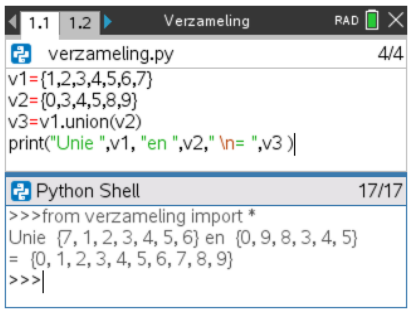
Het data type set is een ongeordende collectie van unieke elementen; m.a.w. wiskundig gezien een verzameling. We laten het aan de lezer over te experimenteren met sets en de methodes beschikbaar voor sets.

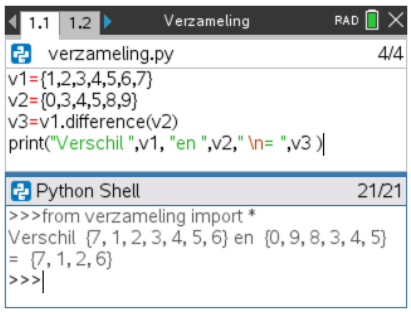












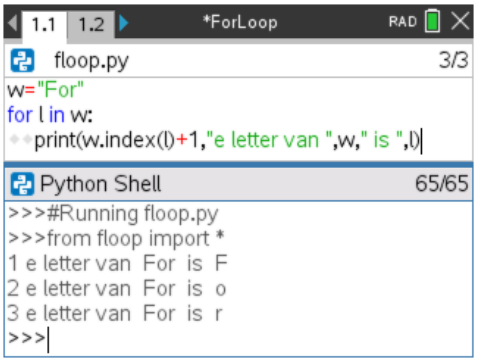
12. For-lus

Lussen worden gebruikt voor het automatisch uit voeren van code of blokken van code. We starten met de For-lus die handelt als een iterator. Het doorloopt items uit een geordende rij, b.v. lijsten, strings en tuples.

De structuur van een for-lus is als volgt:

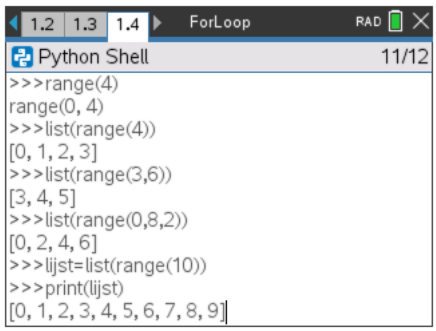
```
for index in object:
    block
```

Twee voorbeelden, baserend op een lijst en een string:



Een operator die vaak gebruikt wordt voor een for-lus is de range()-operator.
De range()-operator heeft de volgende syntax:

- range(4) van 0 tot 4, 4 niet inbegrepen
- range(3,6) van 3 to 6, 6 niet inbegrepen
- range(0,8,2) van 0 tot 8 met stapgrootte 2, 8 niet inbegrepen



range() is een generator!

Een generator genereert data zonder deze data op te slaan in het geheugen.

De combinatie van range() en de list()-functie creëert effectief een lijst.

Voorbeeld 1

Het werpen van een dobbelsteen

De volgende code simuleert het werpen van een dobbelsteen, waarbij voor iedere worp gecheckt wordt of het aantal ogen zes is. In het geval van een zes wordt de teller verhoogd met 1.
Uiteindelijk printen we de benadering van de kans op zes bij het werpen van een dobblesteen: $P(zes) = \frac{1}{6}$.

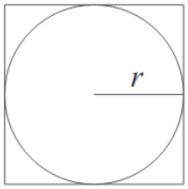
```
from random import *

aantal=int(input("Aantal worpen: "))
teller=0

for i in range(aantal):
    worp=randint(1,6)
    if worp ==6:
        teller+=1

prob=teller/aantal

print("# 6: ",teller,"in", aantal, "worpen")
print("Kans op zes ≈ {0:5.4f}".format(prob))
```

Voorbeeld 2

Monte Carlo benadering voor π

Een Monte Carlo method is een algoritme dat gebruik maakt van random sampling om een probleem op te lossen of te benaderen.

Om π te benaderen tellen we het aantal random gegenereerde punten in een vierkant die binnen de ingeschreven cirkel vallen zoals hiernaast afgebeeld.

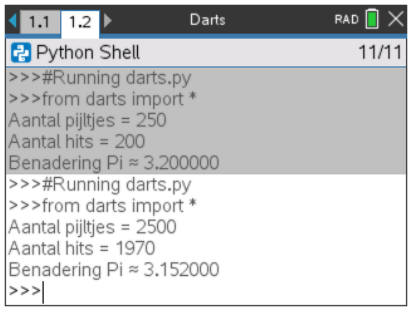


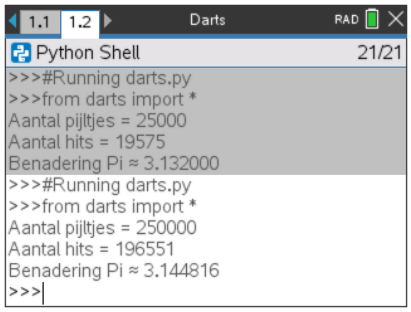
De verhouding van het aantal punten in de cirkel tot het totale aantal gegenereerde punten geeft als volgt een benadering voor π:

$$\frac{N_{\text{cirkel}}}{N_{\text{totaal}}} \approx \frac{\text{Oppervlakte}_{\text{cirkel}}}{\text{Oppervlakte}_{\text{totaal}}} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4} \quad \Rightarrow \pi \approx 4\frac{N_{\text{cirkel}}}{N_{\text{totaal}}}$$

De volgende code simuleert a.h.v. een Monte Carlo methode een benadering van het getal π.

```
from math import *
from random import *

darts=int(input("Aantal pijltjes = "))
hits=0

for worp in range(darts):
    x=random()*2
    y=random()*2
    if sqrt(x**2+y**2)<1:
        hits+=1

prob=4*hits/darts

print("Benadering Pi ≈ {0:5.6f}".format(prob))
```





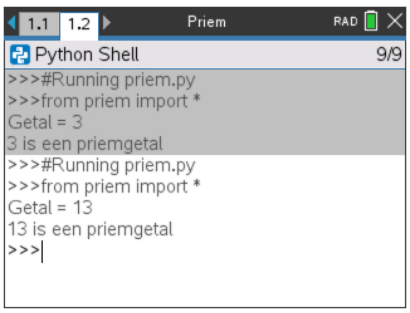
Voorbeeld 3

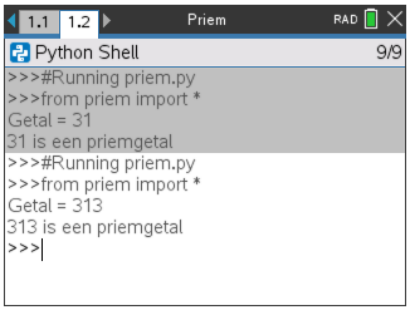
Wanneer is een getal een priemgetal?

Een priemgetal is een geheel getal groter dan 1 dat niet kan geschreven worden als een product van twee kleinere natuurlijke getallen. Priemgetallen worden veel gebruikt in cryptografie, b.v. voor de RSA-code, omdat het ontbinden van grote getallen in een product van priemgetallen niet zo eenvoudig is.

Een methode om te bepalen of een getal n een priemgetal is, is te testen of n een veelvoud is van een van de gehele getallen z met $2 \le z \le \sqrt{n}$.

```
from math import *
n=int(input("Getal = "))
deler=0
if n<=1:
  print("Input ≤ 1 :-(")
else:
  for i in range(2,floor(sqrt(n))+1):
    if n%i == 0:
      deler+=1
  if deler==0:
    print(n,"is een priemgetal")
  else:
    print(n,"is geen priemgetal")
```





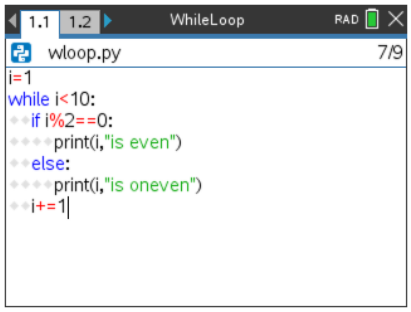
13. While-lus

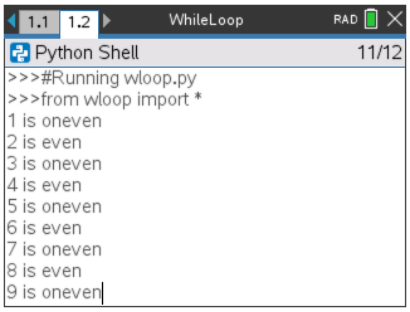
Een while-lus wordt gebruikt voor het uitvoeren van code of blokken van code zolang aan een bepaalde voorwaarde voldaan is.

De structuur van een while-lus is als volgt:

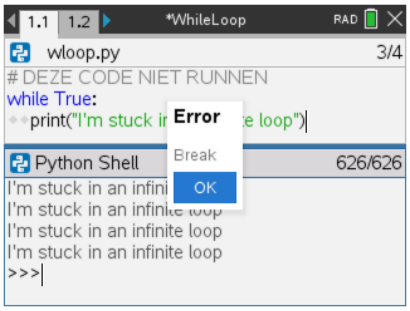
```
while BooleanExpr:
    block
```

Twee eenvoudige voorbeelden voor het illustreren van de syntax van een while-lus:





Wat te doen in het terecht komen in een oneindige loop, b.v. bij het vergeten toe te voegen van i+=1 in bovenstaande voorbeelden?



- Handheld Druk op esc of on
- Windows Druk F12
- MacOs Druk F5

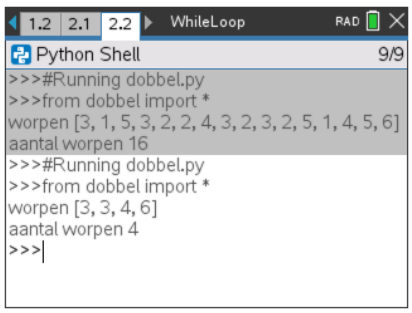
Afhankelijk van het programma, wordt het programma niet altijd onmiddellijk onderbroken; best dan de knop langer ingedrukt te houden

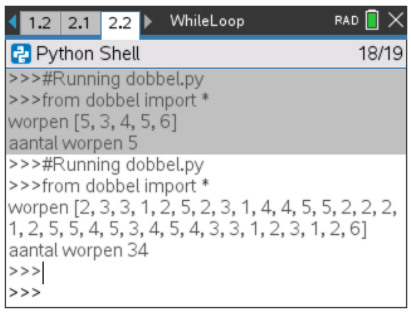
Voorbeeld 1

Het werpen van een dobbelsteen

De volgende code simuleert het werpen van een dobbelsteen tot dat het aantal ogen gelijk is aan zes met als output de ogen van alle worpen en het aantal worpen.

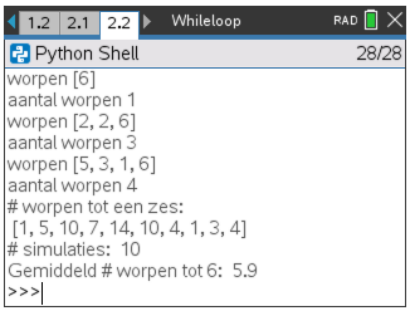
```
from random import *

aantal=0
ogen=0
worpen=[ ]

while ogen != 6:
    ogen=randint(1,6)
    worpen.append(ogen)
    aantal+=1

print("worpen",worpen)
print("aantal worpen",aantal)
```

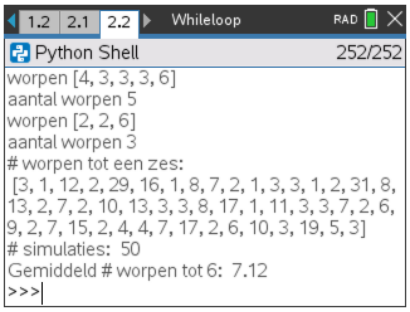


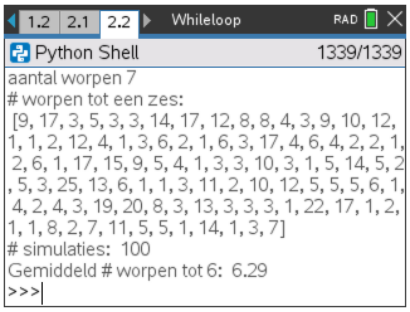


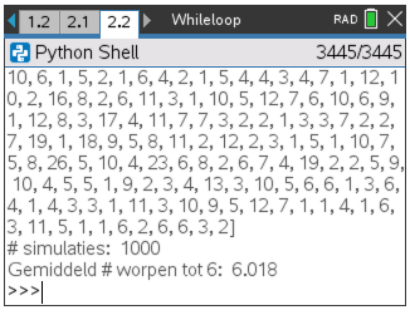
We voegen code toe (for-lus) om deze simulatie een aantal keren na mekaar uit te voeren en telkens het aantal worpen tot zes ogen op te slaan in een lijst. We eindigen het programma met het berekenen van het gemiddelde van deze lijst.

```
from random import *

t=int(input("# experimenten: "))

tot6=[ ]

for n in range(t):
    aantal=0
    ogen=0
    worpen=[ ]

    while ogen != 6:
        aantal=aantal+1
        ogen=randint(1,6)
        worpen.append(ogen)

    print("worpen",worpen)
    print("aantal worpen",aantal)

    tot6.append(aantal)

print("# worpen tot een zes: ",tot6)
print("# simulaties: ",t)

som=0
for i in range(len(tot6)):
    som=som+tot6[i]
print("Gemiddeld # worpen tot zes",som/len(tot6))
```



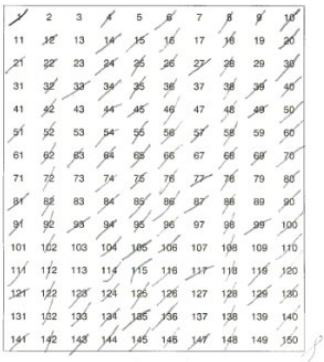






Voorbeeld 2

Zeef van Eratosthenes



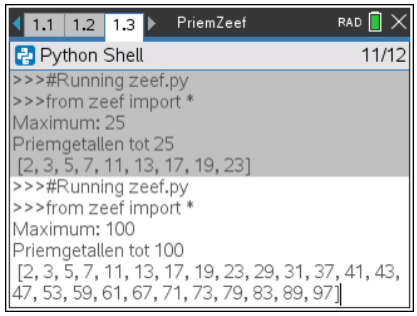
Dit zeer lang gekende algoritme voor het vinden van priemgetallen werkt als volgt:

- Stap 1 Creëer een lijst startend vanaf 2 tot een te kiezen maximum.
- Stap 2 Verwijder alle veelvouden van 2 uit de lijst.
- Stap 3 Kies het kleinste nog overgebleven getal uit de lijst.
- Stap 4 Verwijder alle veelvouden van dit gekozen getal en ga verder met stap 3

Men kan steeds starten met verwijderen van getallen vanaf het kwadraat van het gekozen getal daar alle kleinere veelvouden al verwijderd zijn.

Het algoritme is voltooid als het gekozen getal groter is dan de wortel van het maximum.

```
from math import *
n=int(input("Maximum: "))
priemlijst=[2]
for i in range(3,n+1):
    priemlijst.append(i)

i=2
while i <= sqrt(n):
    if i in priemlijst:
        for j in range(i**2, n+1, i):
            if j in priemlijst:
                priemlijst.remove(j)
    i=i+1

print("Priemgetallen tot",n,"\n",priemlijst)
```

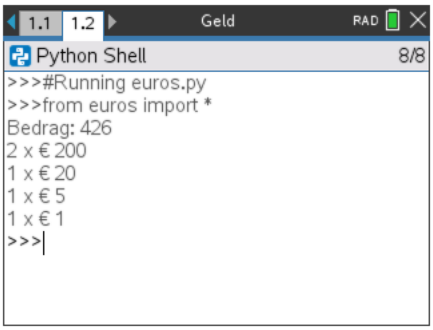


Voorbeeld 3

Verdeling in euros

Het onderstaande programma verdeelt ieder geheel bedrag in € in biljetten van € 200, € 100, € 50, € 20, € 10 of € 5 en munten van € 2 of € 1.

```
bank=[200,100,50,20,10,5,2,1]
bedrag=int(input("Bedrag: "))

for geld in bank:
    aantal=0
    while bedrag>=geld:
        bedrag=bedrag - geld
        aantal=aantal + 1
    if aantal > 0 :
        print (aantal, "x €", geld)
```



We bekijken de code van twee speciale while-lussen.

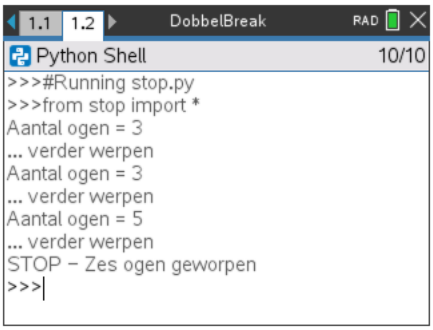
13.1. break & continue

De statements break en continue doen het volgende:

- break breekt uit de dichtstbijzijnde omsluitende lus
- continue gaat naar het begin van de dichtstbijzijnde omsluitende lus

Het onderstaande programma blijft een dobbelsteen werpen simuleren totdat het aantal ogen 6 is:

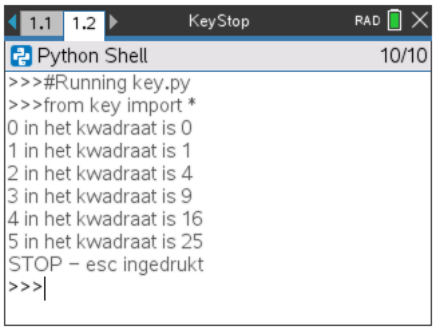
```
from random import *

while True:
    w=randint(1,6)
    if w==6:
        break
    else:
        print("Aantal ogen =",w)
        print("... verder werpen")
        continue

print("STOP - Zes ogen geworpen")
```



13.2. get_key

Gebruikmakend van de module TI System kunnen we een while-lus uitvoeren tot het onderbreken met het intikken van een toets. De syntax ziet er b.v. uit zoals hieronder. De while-lus blijft de blok code uitvoeren tot dat de esc-toets wordt ingedrukt.

```
from ti_system import *
from time import *

i=0

while get_key != "esc":
    print(i,"in het kwadraat is",i**2)
    sleep(1); i+=1

print("STOP - esc ingedrukt")
```



We gebruiken het sleep()-statement van de time-module om het uitvoeren van de code te pauzeren. De output van de kwadraten verschijnt op het scherm per 1 seconde tot het indrukken van de esc-toets.

Merk op dat meerdere code op 1 regel kan geplaatst worden door de code met ; te scheiden.

14. Functies

Functies zijn één van de belangrijkste bouwstenen wanneer we grotere programmeeropdrachten gaan aanpakken. Functies maken het overbodig om code steeds opnieuw te moeten schrijven en maken het mogelijk om blokken van code meer dan één keer te laten uitvoeren.

De Python-syntax van een functie ziet er als volgt uit:

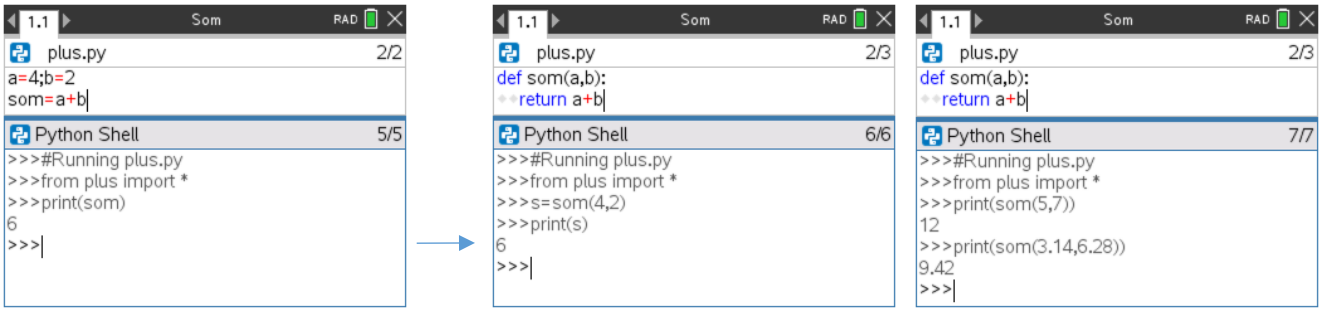
```
def function(argument):
    block
```

We illustreren de syntax met het voorbeeld de som van twee getallen. Het return-statement laat toe een functie een resultaat te bewaren als de waarde van een variabele.

```
a=4
b=2
som=a+b
```

→

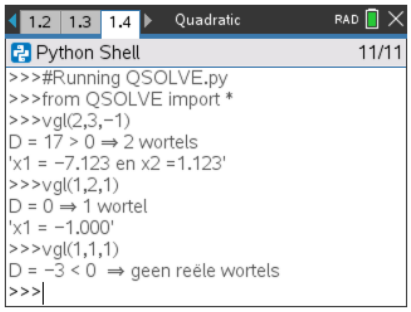
```
def som(a,b):
    return a+b
```

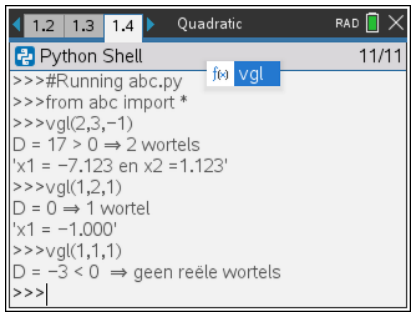


Voorbeeld 1 – De abc-formule

Het oplossen van een vergelijking van de tweede graad $ax^2 + bx + c = 0$ kan als volgt gecodeerd worden:

```
from math import *
def vgl(a,b,c):
    d=b**2-4*a*c
    if d>0:
        print("D =",d,"> 0 ⇒ 2 wortels")
        x1=(-b-sqrt(d))/2*a
        x2=(-b+sqrt(d))/2*a
        return "x1 = {0:1.3f} en x2 ={1:1.3f}".format(x1,x2)
    elif d==0:
        print("D =",d,"⇒ 1 wortel ")
        x1=-b/2*a
        return "x1 = {0:1.3f}".format(r1)
    else:
        print("D =",d,"< 0  ⇒ geen reële wortels")
```





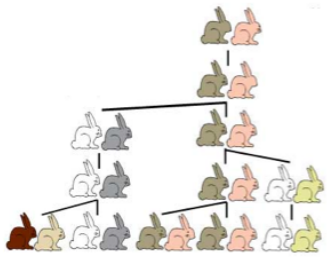
In plaats van het intikken van een gedefinieerde functie, kan de functie ook opgeroepen worden met behulp van de var-knop.

Voorbeeld 2 – Fibonacci & De Gulden Snede

Leonard van Pisa, beter gekend als Fibonacci (= zoon van Bonaccio), was één van de grote wiskundigen van de Middeleeuwen. Alhoewel Fibonacci in Italië geboren was, verbleef hij een hele tijd in Algerije waar zijn vader tewerkgesteld was in een handelsmaatschappij.

Zijn eerste wiskundige stappen zette hij onder begeleiding van Islamitische leraars. Al snel merkte hij de superioriteit van het Arabische talstelsel, met een positionele schrijfwijze en het getal nul, t.o.v. het Romeinse talstelsel. In 1202 voltooide hij zijn bekendste werk: *Liber Abaci*(= Boek van de Abacus of Rekenkunde) waarin hij de Arabische rekenbeginselen in de Westerse wereld introduceerde.

In Liber Abaci formuleerde Fibonacci ook het volgende probleem.



De konijnen van Fibonacci
We plaatsen een paar jonge konijntjes binnen een omheining. Veronderstel dat ze zich na 1 maand kunnen voortplanten en dat ze op het einde van iedere daaropvolgende maand een nieuw paar konijntjes (één mannetje en één vrouwtje) voortbrengen. Veronderstel ook dat ieder baby-paar 2 maanden na hun geboorte zo'n paartje voortbrengt en dat er geen konijntjes doodgaan.

Hoeveel paren zijn er na 12 maanden?

Als we iedere maand de paren konijnen tellen, krijgen we de volgende rij getallen – de rij van Fibonacci:

1,1,2,3,5,8,13,21,34,55,89, 144, …

Vanaf het derde getal van deze rij geldt dat ieder getal de som van de twee voorgaande termen is. We zien dat er na 12 maanden 144 konijnenparen zijn.

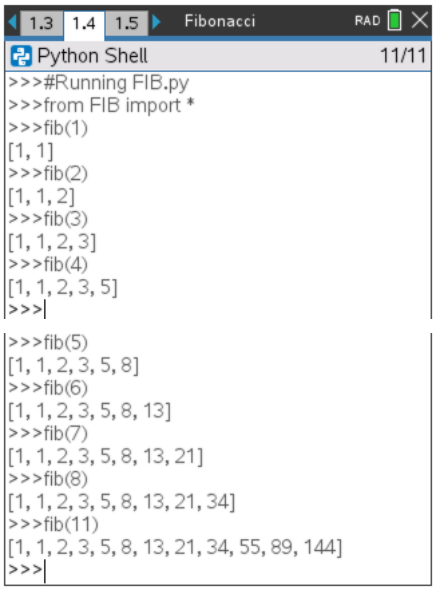
De rij van Fibonacci wordt als volgt op gebouwd:

$F_0 = 1$
$F_1 = 1$
$F_n = F_{n-1} + F_{n-2}$ voor $n > 1$

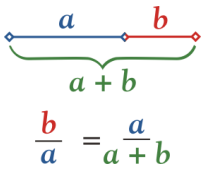
Men kan bewijzen dat: $F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$.

Het volgende programma berekent de rij van Fibonacci:

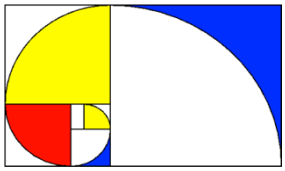
```
def fib(n):
    a=1
    b=1
    fibseq=[a,b]
    for i in range (n-1):
        a,b=b,a+b
        fibseq.append(b)
    return fibseq
```



De Gulden Snede
De gulden snede is de verdeling van lijnstuk in twee delen waarbij het grootse deel zich verhoudt tot het kleinste, als het lijnstuk tot het grootste.



$$\frac{b}{a} = \frac{a}{a+b}$$



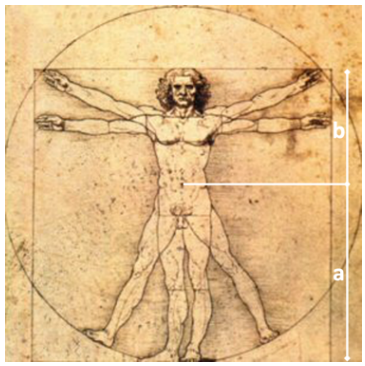
Deze verhouding wordt ook wel het gouden getal genoemd:

$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618\ldots$

De gulden snede komt veelvuldig voor in de natuur, architectuur, kunst, …

De gulden gnede kan als volgt geschreven worden als een kettingbreuk:

$$\varphi = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cdots}}}}}$$



We bekijken enkele benaderingen van $\varphi = 1.618\ldots$ m.b.v. deze kettingbreuk:

$$\varphi \approx 1 + 1 = 2 = \frac{2}{1} = 2$$

$$\varphi \approx 1 + \frac{1}{1+1} = \frac{3}{2} = 1{,}5$$

$$\varphi \approx 1 + \frac{1}{1+\frac{1}{1+1}} = \frac{5}{3} = 1{,}666$$

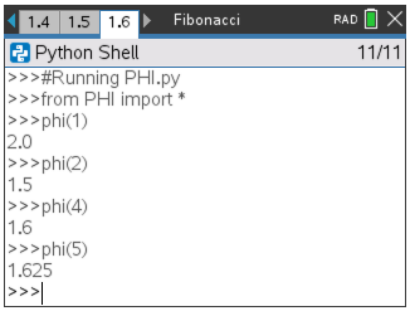
$$\varphi \approx 1 + \frac{1}{1+\frac{1}{1+\frac{1}{1+1}}} = \frac{8}{5} = 1.6$$

$$\varphi \approx 1 + \frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+1}}}} = \frac{13}{8} = 1.625$$

Dit geeft dat we φ kunnen benaderen door de verhouding van twee opeenvolgende Fibonacci-getallen: $\varphi = \lim_{n\to\infty} \frac{F_{n+1}}{F_n}$.

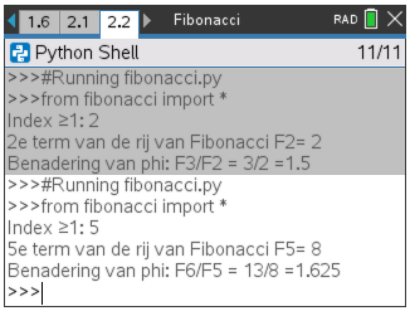
Voor het benaderen van φ door de verhouding van twee opeenvolgende Fibonacci getallen gebruiken we zojuist gedefinieerde functie fib() uit het FIB.py programma. We starten de code met from FIB import *.

```
from FIB import *
def phi(n):
    return fib(n+2)[n+1]/fib(n+1)[n]
```



Vanzelfsprekend kunnen beide definities
Samengevoegd worden in één programma.

```
# Rij van Fibonacci
def fib(n):
    a=1
    b=1
    fibseq=[a,b]
    for i in range (n-1):
        a,b=b,a+b
        fibseq.append(b)
    return fibseq

# Benadering van het gouden getal
def phi(n):
    return fib(n+2)[n+1]/fib(n+1)[n]

n=int(input("Index ≥1: "))

print("{i}e term van de rij van Fibonacci F{i}=".format(i=n),fib(n)[n])

print("Benadering van phi: F{}/F{} = {}/{} ={}".format(n+1,n,fib(n+1)[n+1],fib(n)[n],phi(n)))
```

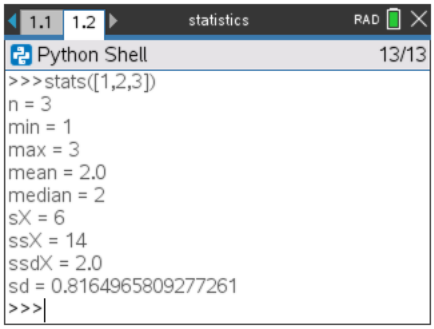


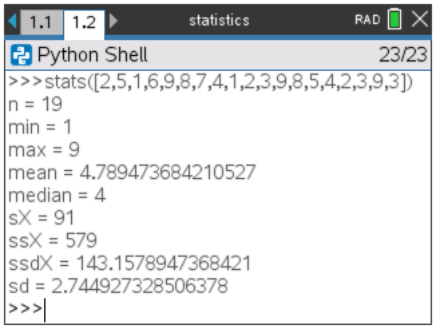
Voorbeeld 3 – Statistische kengetallen

Hieronder de code, in de vorm van definities, voor het berekenen van enkele statistische kengetallen van een dataset (lijst). min() en max() zijn ingebouwde functies.

```
def sx(list):
    total=sum(list)
    return total

def ssx(list):
    total=0
    for i in list:
        total+=i**2
    return total

def mean(list):
    n=len(list)
    total=sum(list)
    return total/n

def ssdx(list):
    total=0
    av=mean(list)
    for i in list:
        total+=(i-av)**2
    return total

def sd(list):
    return (ssdx(list)/len(list))**0.5
```

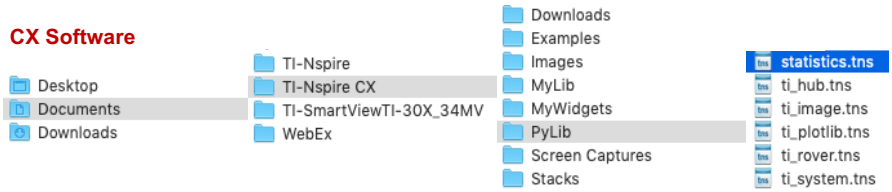
```
def median(list):
    slist=sorted(list)
    if len(list)%2:
        mid=int(len(list)/2)+1
        return slist[mid-1]
    else:
        mid=int(len(list)/2)
        return (slist[mid]+slist[mid-1])/2

def stats(list):
    print("n =", len(list))
    print("min =", min(list))
      print("max =",max(list))
    print("mean =",mean(list))
    print("median =", median(list))
    print("sX =",sx(list))
    print("ssX =",ssx(list))
    print("ssdX =",ssdx(list))
    print("sd =",sd(list))
    return
```



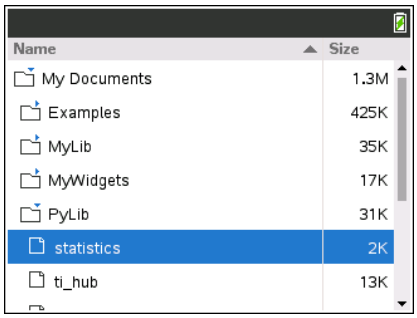


Om deze functies te gebruiken als een zelf gedefinieerde module, plaats je het tns-document met het programma met de functies in de PyLib-folder (... > Documents > TI-Nspire CX > PyLib).

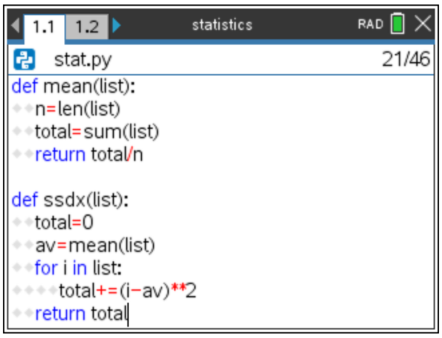
CX Software



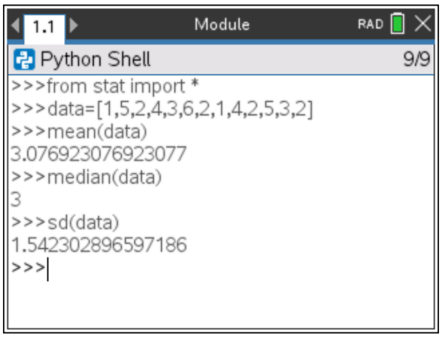
CX II-T Handheld



De statistische functies gedefinieerd in het document statistics.tns kunnen nu gebruikt worden na een import van het betreffende programma stat.py in ieder document:



stat-functies → in doc Module



15. Recursie

Een eenvoudige omschrijving van een recursieve functie is een functie die optreedt als onderdeel van zichzelf. M.a.w. een recursieve functie is een functie die zichzelf aanroept. Vanzelfsprekend heeft recursie meer gecompliceerde definities en toepassingen in de informatica en het programmeren.

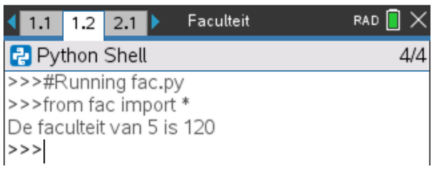
Met behulp van recursieve functies kan het probleem worden onderverdeeld in sub-problemen die eenvoudig kunnen worden opgelost. Maar het is soms moeilijker om de logica van recursieve functie te volgen.

Voorbeeld 1 – Faculteit

Voor ieder natuurlijk getal $n \geq 1$ geldt: $n! = \prod_{k=1}^{n} k = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$
Eigenschap: $n! = n \cdot (n-1)!$

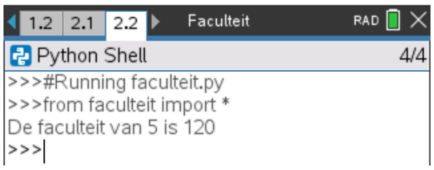
Met de definitie

```
def fac(n):
♦♦uitkomst = 1
♦♦for i in range(1,n+1):
♦♦♦♦uitkomst = uitkomst * I
♦♦return uitkomst

a=5
print("De faculteit van",a,"is",fac(a))
```



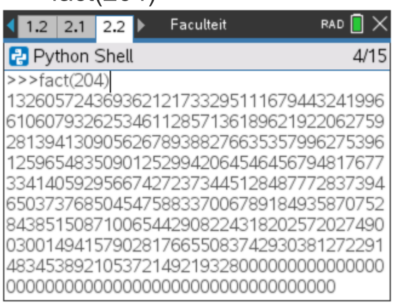
Met recursie

```
def fact(n):
♦♦if n==1:
♦♦♦♦return 1
♦♦else:
♦♦♦♦return (n*fact(n-1))

a=5
print("De faculteit van",a,"is",fact(a))
```

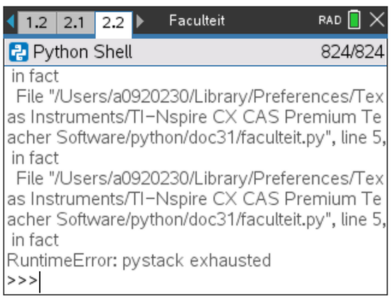


Python legt recursie een beperking op omdat elke keer een functie zichzelf aanroept, tussenliggende waarden bewaard moeten worden in het geheugen. Voor het bovenstaande programma kan hoogstens 204! berekend worden.

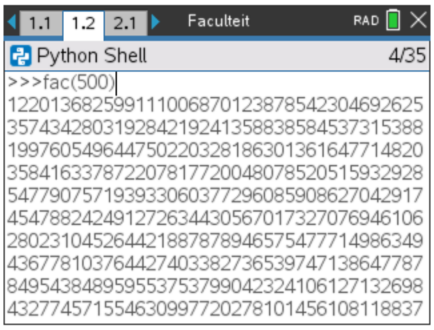
fact(204)

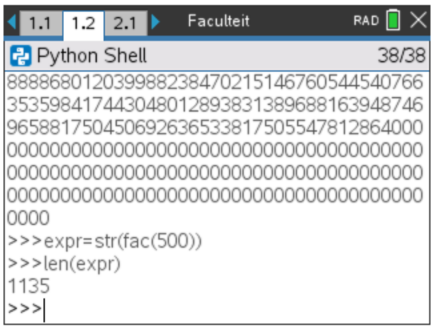


fact(205)



Voor het programma gebaseerd op de definitie stelt zich dit probleem niet. Indien we 500! omzetten naar een string (met het str()-statement) kunnen we het aantal cijfers van 500! bepalen.



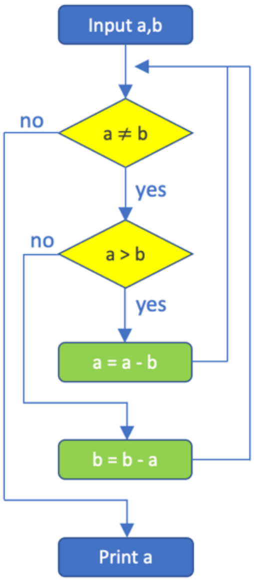


Voorbeeld 2 – GGD

De grootste gemeenschappelijke deler, ggd, van twee gehele getallen is het grootste positieve geheel getal waardoor deze getallen deelbaar zijn.

Algoritme van Euclides
Zolang dat a ≠ b:

1. Bepaal grootste van beide getallen
2. Vervang het grootste getal door het verschil van het grootste met het kleinste
3. Ga naar stap 1 met als getallen dit verschil en het kleinste getal

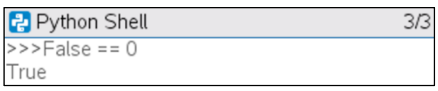



```
a=int(input("a= "))
b=int(input("b= "))
x,y = a,b
while x!=y:
  if x>y:
    x=x-y
  else:
    y=y-x
print("De grootste gemene deler van {} en {} = {}".format(a,b,x))
```

GGD m.b.v. modulo-rekenen

```
a=int(input("a= "))
b=int(input("b= "))
x,y = a,b
while y!=0:
  x,y=y,x%y
print("De grootste gemene deler van {} en {} = {}".format(a,b,x))
```

Merk op dat while y!=0: kan vervangen worden door while y: .



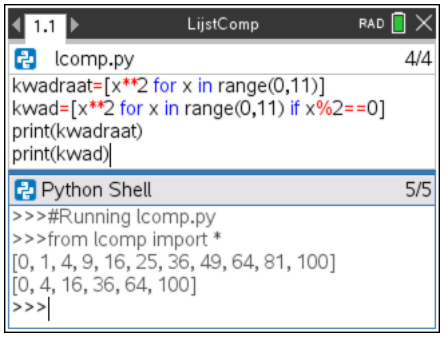
GGD recursief geprogrammeerd

```
a=int(input("a= "))
b=int (input("b=" ))

def ggd(p,q):
    if q==0:
        return(p)
    else:
        return ggd(q,p%q)

print("De grootste gemene deler van {} en {} = {}".format(a,b,ggd(a,b)))
```

16. Lijstcomprehensie

Lijstcomprehensie (sequentie-besef) is een geavanceerde constructie om lijsten te genereren. Het is gebaseerd op de wiskundige notatie om verzamelingen te definiëren:

$S = \{x^2 \mid x \in \mathbb{N}, x \leq 10\} = \{0,1,4,9,16,25,49,64,81,100\}$.



D.m.v. van lijstcomprehensie declareren we als volgt een lijst met de bovenstaande elementen:

```
kwadraat=[x**2 for x in range(0,11)].
```

Men kan deze syntax ook gebruiken in combinatie met een if-statement:

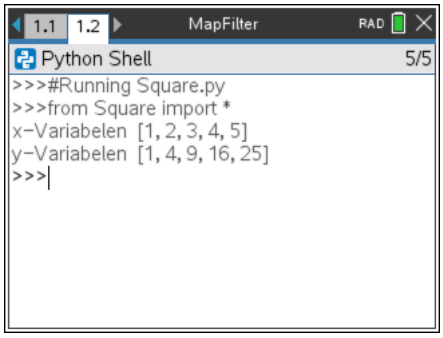
```
kwadraat=[x**2 for x in range(0,11) if x%2==0].
```

17. Map & filter

Python heeft ook de ingebouwde functies map() en filter() waarmee eenzelfde functie kan uitgevoerd worden op iedere item van een itereerbaar object, b.v. lijsten, strings, ...

We definiëren de functie kwadraat

```
def kwadraat(x):
    return x**2
```



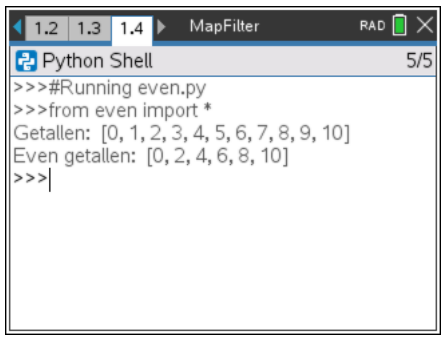
We passen deze functie toe op de lijst [1,2,3,4,5]. Om een nieuwe lijst te creëren moet het map()-statement in combinatie met list() gebruikt worden.

```
xvar=[1,2,3,4,5]
yvar=list(map(kwadraat,xvar))
print("x-Variabelen: ",xvar)
print("y-Variablelen: ",yvar)
```

Met het filter()-statement kan een keuze gemaakt worden uit items van een itereerbaar object, items waarvoor een bepaalde functie waar is.

De onderstaande functie checkt of een getal even is:

```
def even(x):
    return x%2==0
```



En met het filter()-statement selecteren we de even getallen uit de lijst getal=[0,1,2,3,4,5,6,7,8,9,10]:

```
getal=[n for n in range(0,11)]

even_getal=list(filter(even,getal))

print("Getallen: ",getal)
print("Even getallen: ",even_getal)
```

18. Lokaal versus Globaal

De eerste keer dat de waarde van x geprint wordt, wordt gebruik gemaakt van de declaratie van x in het hoofdblok van de code: x=5.

We veranderen de waarde van x, x=2, lokaal in de functie. Wanneer we de waarde van x veranderen, heeft dit geen effect op de waarde toegekend in het hoofdblok. Dit toont het laatste print()-statement.

Men kan ook in b.v. een functie een globale variabele definiëren met het global-statement als volgt:

Merk op dat door x globaal te maken in de functie func, x niet kan gebruikt worden als argument van de functie. In het eerdere voorbeeld, func(x), is het nodig om x als argument te gebruiken, anders krijg je de error-boodschap: *NameError: local variable referenced before assignment*.

Nog twee programma's om het verschil tussen lokaal en globaal te illustreren:

For-lus

```
p=3.14
print("p = ",p)
lijst=[ ]
for p in range(0,3):
  lijst.append(p)
print("Lijst = ",lijst)
print("p = ",p)
```

p globaal

```
Python Shell
>>>#Running gvar.py
>>>from gvar import *
p =  3.14
Lijst =  [0, 1, 2]
p =  2
>>>
```

Comprehensie

```
p=3.14
print("p = ",p)
lijst=[p for p in range(0,3)]
print("Lijst = ",lijst)
print("p = ",p)
```

p lokaal

```
Python Shell
>>>#Running lvar.py
>>>from lvar import *
p =  3.14
Lijst =  [0, 1, 2]
p =  3.14
>>>
```

19. Numerieke methodes

19.1. Iteratieve banen

iteratie

ite·ra·tie (niet: **i·teratie**)
/itər'a(t)si/, /iter'a(t)si/
de (v.)
‹1650› ‹Fr. *itération*

1 ▪ herhaling

2 ▪ het herhalen van een wiskundige bewerking

3 ▪ herhaling als stijlfiguur

We bekijken het iteratief-proces waarbij de verwerking gebeurt met een (reële) functie en een startwaarde x_0:

$$x_0 \underset{F}{\rightarrow} x_1 = F(x_0) \underset{F}{\rightarrow} x_2 = F(x_1) = F(F(x_0)) = F^2(x_0) \underset{F}{\rightarrow} .... \underset{F}{\rightarrow} x_n = F(x_{n-1}) = F^n(x_0) \underset{F}{\rightarrow} .....$$

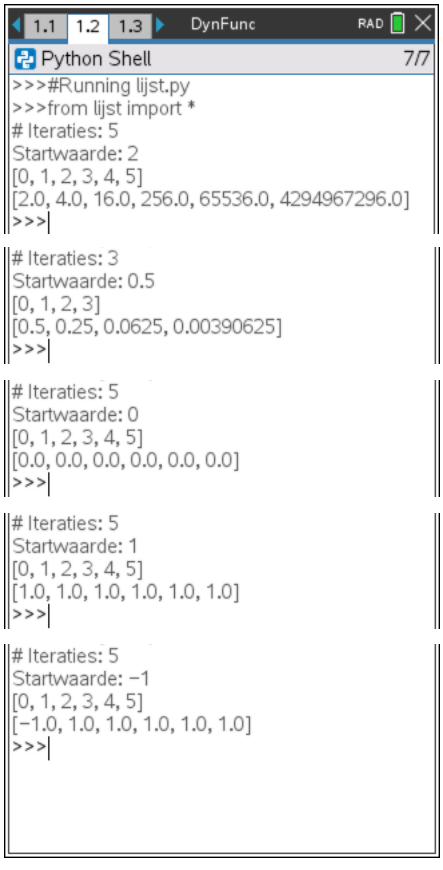
Waarbij F^n betekent dat F n-keer wordt uitgevoerd. Deze functie F noemt men de iteratiefunctie.

Voor een startwaarde x_0 genereert een iteratieproces een rij $x_0, x_1, x_2, x_3, ..., x_n, ...$.

Deze rij noemt men de baan behorende bij de startwaarde x_0.

We schrijven een programma dat de banen berekent voor $F(x) = x^2$ en vervolgens visueel/grafisch voorstelt.

```
def f(x):
    return x**2

n=int(input("# Iteraties: "))
x0=float(input("Startwaarde: "))

index=[i for i in range(0,n+1)]
iteratie=[x0]

for i in index:
    iteratie.append(f(iteratie[i]))
iteratie.pop()

print(index)
print(iteratie)
```



We kunnen het volgende gedrag afleiden:

Als $|x|>1$ zal $F^n(x)$ steeds groter en groter worden.

We noteren dit als volgt: $F^n(x) \rightarrow +\infty$ voor $n \rightarrow +\infty$.

Als $0<|x|<1$ komt $F^n(x)$ steeds dichter bij 0.

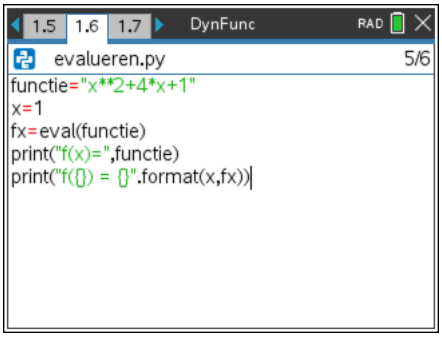
We noteren dit als volgt: $F^n(x) \rightarrow 0$ voor $n \rightarrow +\infty$.

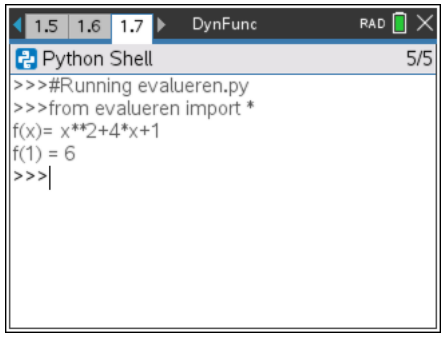
Enkele speciale banen:

$x_0 = 1 \Rightarrow \forall n : F^n(1) = 1$ en $x_0 = -1 \Rightarrow \forall n \geq 1 : F^n(1) = 1$

$x_0 = 0 \Rightarrow \forall n : F^n(0) = 0$

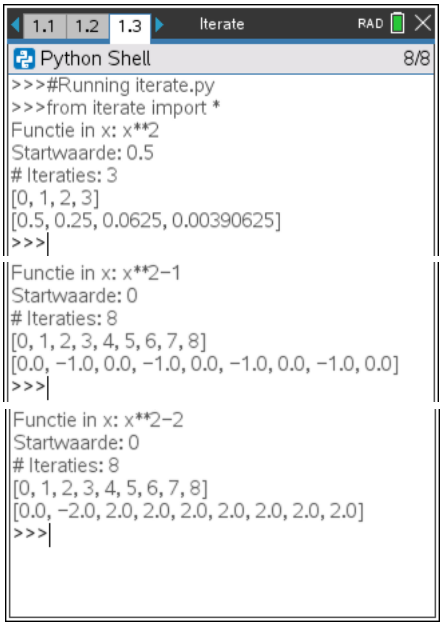
We maken a.h.v. deze code een functie iterate() met als argumenten de functie, de startwaarde en het aantal iteraties. Hiervoor gebruiken o.a. het eval()-statement. Een verschil tussen MicroPython en Python is dat in MicroPython eval() geen gebruik kan maken van lokale variabelen.





De iterate()-functie ziet er als volgt uit:

```
def iterate(fx,x0,n):
    global x
    iterlst=[x0]
    for i in range(0,n+1):
        x=iterlst[i]
        iterlst.append(eval(fx))
    iterlst.pop()
    return iterlst

iterator=input("Functie in x: ")
start=float(input("Startwaarde: "))
aantal=int(input("# Iteraties: "))

index=[i for i in range(0,aantal+1)]
iteratie=iterate(iterator,start,aantal)

print(index)
print(iteratie)
```

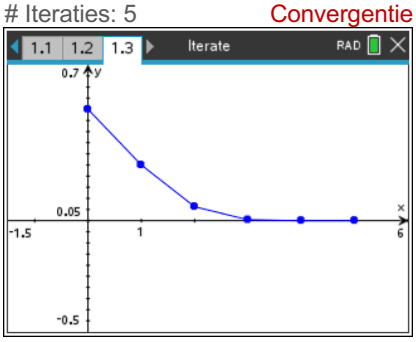


Gebruikmakend van het store_list()-statement van de TI System-module kunnen we de iteratie visueel voorstellen als een scatter plot in Graphs

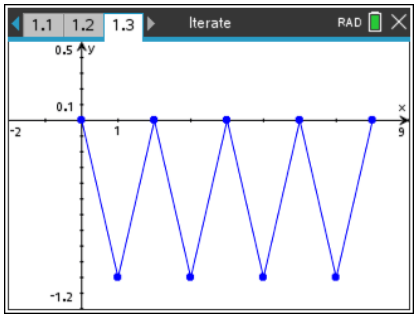
```
from ti_system import *
……
store_list("index",index)
store_list("iteratie",iteratie)
```

Enkele voorbeelden

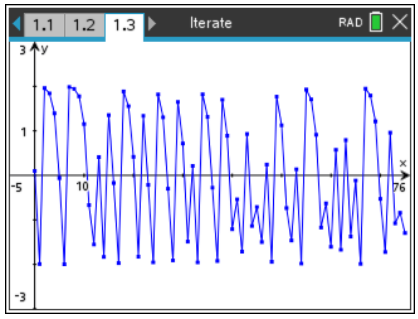
Functie in x: x**2
Startwaarde: 0.5
Iteraties: 5
Convergentie



Functie in x: x**2-1
Startwaarde: 0
Iteraties: 8
Periodiek



Functie in x: x**2-2
Startwaarde: 0.1
Iteraties: 75
Chaos



Nog twee definities:

- Fixpunt of vast punt

  x is een vast punt van F als $F(x)=x$.

  $x \to F(x)=x \to F^2(x)=F(F(x))=F(x)=x \to x \to x \to x \to ...$

  We zeggen dat de startwaarde ter plaatse blijft.

  Het bepalen van een fixpunt komt neer op het oplossen van de vergelijking $F(x)=x$.

- Periodiek punt

  x noemen we een periodiek punt als er een $n>0$ bestaat zodat $F^n(x)=x$.

  $x_0 \to x_1=F(x_0) \to x_2=F^2(x_0) \to x_3=F^3(x_0) \to ... \to x_n=F^n(x_0)=x_0 \to x_1 \to x_2 \to ...$

  $x_0 \to x_1 \to x_2 \to x_3 \to ... \to x_{n-1} \to x_0 \to x_1 \to x_2 \to x_3 \to ... \to x_{n-1} \to x_0 \to ...$

  De baan van x_0 is een periodieke baan met periode n of een n-cyclus.

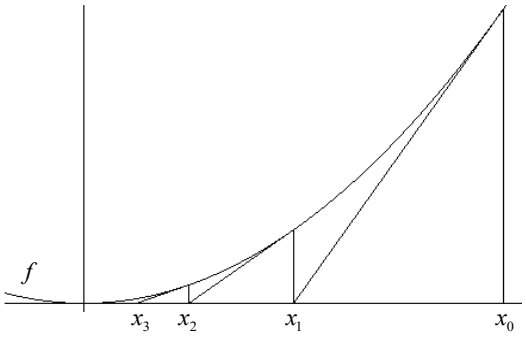
19.2. Newton-Raphson

De methode van Newton-Raphson is een iteratieve numerieke benaderingsmethode van nulpunten die gebruik maakt van de raaklijn aan de grafiek van de functie.

Vertrekkende van een startwaarde x_0 berekenen we het snijpunt van de raaklijn aan de grafiek in het punt $(x_0, f(x_0))$ met de x-as.

De *x*-coördinaat van dit snijpunt is de volgende benadering, x_1, van het nulpunt. We herhalen deze werkwijze voor x_1 en bekomen zo het punt x_2 als snijpunt van de raaklijn in het punt $(x_1, f(x_1))$ met de x-as.

Het steeds verder zetten van dit proces genereert een rij $x_0, x_1, x_2, x_3, x_4, ...$ die voor heel wat functies convergeert naar een nulpunt. De exacte voorwaarde voor convergentie behandelen we hier niet.



We bepalen de iteratiefunctie die aan de basis ligt van de methode van Newton-Raphson.

De vergelijking van de raaklijn aan de grafiek van f in $(x_n, f(x_n))$ is van de vorm:

$$y-f(x_n)=f'(x_n)(x-x_n) \Leftrightarrow y=f(x_n)+f'(x_n)(x-x_n).$$

Voor het snijpunt van de raaklijn met de x-as geldt $y=0$ zodat $x=x_n-\frac{f(x_n)}{f'(x_n)}$.

De iteratiefunctie is $N(x)=x-\frac{f(x)}{f'(x)}$ en genereert de volgende rij:

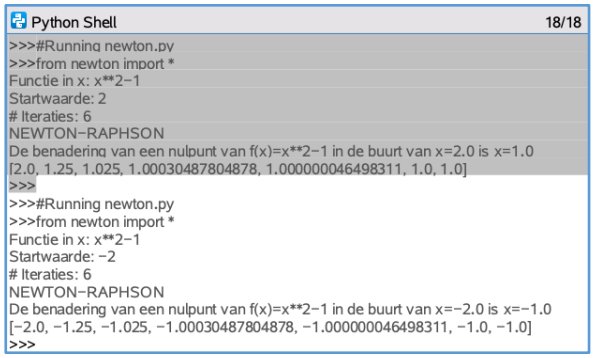
$$x_0,\quad x_1=x_0-\frac{f(x_0)}{f'(x_0)},\quad x_2=x_1-\frac{f(x_1)}{f'(x_1)},\quad x_3=x_2-\frac{f(x_2)}{f'(x_2)},\quad x_4=x_3-\frac{f(x_3)}{f'(x_3)},...$$

Een code voor de methode van Newton-Raphson:

```
def iterate(fx,x0,n):
    global x
    iterlst=[x0]

    for i in range(0,n+1):
#     Bepalen numerieke afgeleide
        e=0.001
        x=iterlst[i]+0.001
        y2=eval(fx)
        x=iterlst[i]-0.001
        y1=eval(fx)
        df=round((y2-y1)/(2*e),5)
#     Bepalen volgend punt iteratieproces
        x=iterlst[i]
        iterlst.append(iter[i]-(eval(fx))/df)
    iterlst.pop()
    return iterlst

functie=input("Functie in x: ")
start=float(input("Startwaarde: "))
aantal=int(input("# Iteraties: "))

iteratie=iterate(functie,start,aantal)
nulpunt=iteratie[len(iteratie)-1]

print("NEWTON-RAPHSON")
print("De benadering van een nulpunt van f(x)={} in de buurt van x={} is
x={}".format(functie,start,nulpunt))
print(iteratie)
```

$$f'(x_0) \approx \frac{f(x_0+\varepsilon)-f(x_0-\varepsilon)}{2\varepsilon} \quad \text{met} \quad \varepsilon = 0.001$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



Zoals al aangegeven behandelen we de exacte voorwaarde voor convergentie hier niet. De bovenstaande code convergeert niet altijd en kan zelfs leiden tot een error, b.v. *ZeroDivisionError: divide by zero*. Hou er ook rekening mee dat het hier over een numerieke benadering gaat voor de afgeleide en dat de grootte van ε (of e in de code) een rol speelt.

Programmeeropdrachten

Opdracht 1: $\pi \approx \cdots$

Leibniz was de eerste die in 1674 een reeksontwikkelingen als benadering van π formuleerde:

$$\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

a. Definieer een functie die de term van de reeks bepaalt in functie van n.

b. Schrijf een programma dat π benadert d.m.v. de eerste 500000 termen van de reeks.

Opdracht 2: Hoeveel nullen?

De onderstaande functie telt het aantal nullen dat voorkomt in een getal.
Het str()-statement zet een getal om in een string: b.v. str(25) resulteert in “25” en str(2.5) in “2.5”.

```
def aantal_nullen(a):
    cijfers=str(a)
    aantal=0
    for c in cijfers:
        if c == "0":
            aantal += 1
    return aantal
```

a. Bereken met deze functie het aantal nullen in 100! .

b. Schrijf een programma dat het kleinste getal n berekent waarvoor geldt dat n! minstens 100 nullen heeft.

Opdracht 3: Palindroom

Een palindroom is een woord waarin de letters symmetrisch gerangschikt zijn, zodanig dat het woord van achter naar voren gelezen hetzelfde is als van voor naar achter.

a. Definieer een functie met als argument een woord (string) die als resultaat True geeft indien het woord een palindroom is en False indien niet.

Een palindroomgetal of numeriek palindroom is een natuurlijk getal dat hetzelfde blijft wanneer zijn cijfers in omgekeerde volgorde worden geschreven, b.v. 13831.

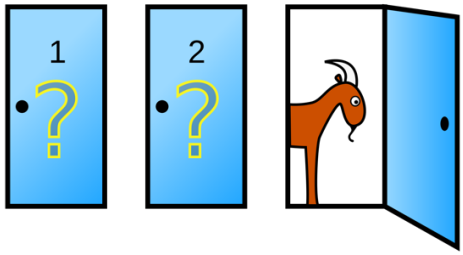
b. Schrijf een programma, gebruikmakend van deze functie, dat alle palindroomgetallen afdrukt tussen 2000 en 3000.

Opdracht 4: Het driedeurenprobleem

In een quiz wordt een deelnemer geconfronteerd met drie gesloten deuren.

Achter één van de deuren staat een auto en achter de twee andere twee deuren een geit. De deelnemer mag een deur aanwijzen en krijgt als prijs datgene wat zich achter die deur bevindt.

Als de deelnemer een deur heeft aangewezen, opent de presentator een van de andere deuren waarachter een geit staat. De presentator geeft de deelnemer daarna de mogelijkheid om te wisselen van gesloten deur, m.a.w. om in plaats van de eerst gekozen deur een andere nog gesloten deur te kiezen.



Wat moet de deelnemer doen? Kan hij beter wisselen van deur, of maakt het niets uit?

Is de kans op het winnen van de auto groter als de deelnemer van deur wisselt?

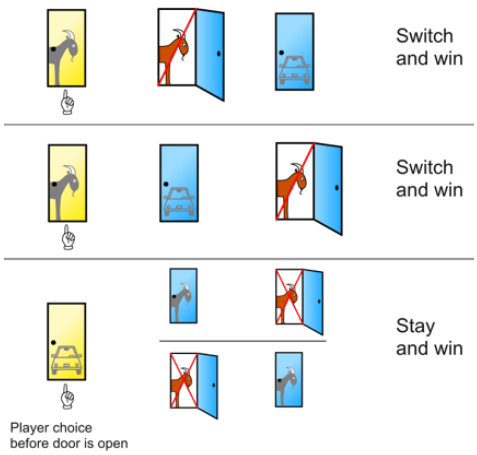
Origineel heet deze brain teaser The Monty Hall problem, gebaseerd op de Amerikaanse TV show Let's make a chance met als gastheer Monty Hall. Het probleem werd voorgelegd aan The America Statistician (een wetenschappelijke academische magazine) in 1975.

Veronderstel dat de deelnemer deur 1 kiest. Dan zijn er de volgende mogelijkheden:

Deur 1	Deur 2	Deur 3	Resultaat bij keuze 1 blijven	Resultaat veranderen van keuze
Geit	Geit	**Auto**	Deelnemer wint een geit	**Deelnemer wint een auto**
Geit	**Auto**	Geit	Deelnemer wint een geit	**Deelnemer wint een auto**
Auto	Geit	Geit	**Deelnemer wint een auto**	Deelnemer wint een geit

Indien de deelnemer bij zijn keuze blijft, is de kans op de auto $\frac{1}{3}$.

En indien de deelnemer zijn keuze wijzigt, is de kans op de auto $\frac{2}{3}$.



In de onderstaande definitie is de waarde van het argument switch gelijk aan True of False.

```
from random import *
def game(switch):
    # index voor prijs
    prijs=randint(0,2)
    # index voor keuze
    keuze=randint(0,2)
    # test resultaat keuze
    resultaat=keuze==prijs
    if switch:
        return not resultaat
    else:
        return resultaat
```

a. Ga na dat de bovenstaande functie het driedeurenprobleem simuleert en dat de keuze van de deelnemer effectief afhankelijk is van de waarde van het argument switch.

b. Gebruik deze functie om de spelsituatie 10000 keer te simuleren en bepaal hiermee de kans op een auto bij het niet wisselen van keuze en bij het wisselen van keuze.

Opdracht 5: Recursie

a. Schrijf een programma dat recursief de n^e (gehele n ≥ 0) macht van een getal a berekent.

b. Benader $\sqrt{2}$ recursief gebruikmakend van de kettingbreuk:

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{\ldots}}}}$$

Opdracht 6: Iteratie

Schrijf een programma dat $\sqrt{2}$ benadert d.m.v. de Babylonische methode, een veelgebruikte methode die gebruikt wordt door computer en rekenmachines.

Deze methode is gebaseerd op de gelijkheid: $\sqrt{2} = \frac{1}{2}\left(\sqrt{2} + \frac{2}{\sqrt{2}}\right)$.

Vertrekkende van een startwaarde $a_0 = 1$, een ver van nauwkeurige benadering van $\sqrt{2}$, wordt $\sqrt{2}$ stap voor stap iteratief beter benaderd als volgt:

$$a_{n+1} = \frac{1}{2}\left(a_n + \frac{2}{a_n}\right) = \frac{a_n}{2} + \frac{1}{a_n}$$

Voor $a > 0$ de limiet van dit proces geldt: $a = \frac{a}{2} + \frac{1}{a}$, m.a.w $a^2 = 2$.

Verdieping

a. Error-boodschappen

De python error-boodschappen kunnen gebruikt worden om de invoer te specifiëren.

Drie voorbeelden van error-boodschappen:

1. NameError: het gebruik van een niet gedefinieerde variabele, b.v. in een print()-statement

```
test1.py
print(x)

Python Shell
NameError: name 'x' isn't defined
```

2. ValueError: het uitvoeren van een functie op een argument van het verkeerde type

```
test2.py
int("Python")

Python Shell
ValueError: invalid syntax for integer with base 1 0: 'Python'
```

3. TypeError: het uitvoeren van een operatie op het verkeerde type

```
test3.py
w="Python"
w+=1

Python Shell
TypeError: can't convert 'int' object to str implicitly
```

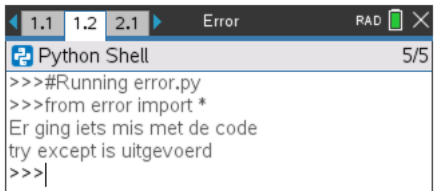
b. try & except

De statements try en except doen het volgende:

- try error-code testen
- except uitvoeren van code in geval van error

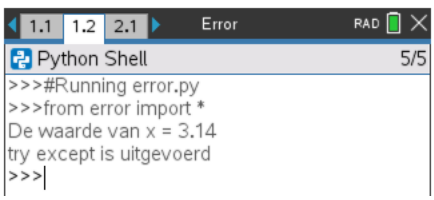
We verduidelijken even deze structuur. Bij het runnen van de onderstaande code krijg je geen error-melding.
try: probeert de code uit te voeren en in het geval van een error wordt de code in het except:-blok uitgevoerd.

```
try:
    print("De waarde van x =",x)
Except:
    print("Er ging iets mis met de code")
print("try except is uitgevoerd")
```



Indien er geen error optreedt, m.a.w. de variabele x is gedeclareerd, wordt de code in het try:-blok uitgevoerd

```
x=3.14
try:
    print("De waarde van x =",x)
Except:
    print("Er ging iets mis met de code")
print("try except is uitgevoerd")
```



Voor het except-statement kan het error-type gespecifieerd worden, b.v. except NameError: voor bovenstaande code.

c. Input op maat

Met de statements try en except kunnen we de input specifiëren en vermijden dat we een error-boodschap krijgen bij hier invoeren van een verkeerd type.

In het volgende programma willen we de input beperken tot een geheel getal n tussen 0 en 5, $0 \leq n \leq 5$, en blijven vragen naar input tot de ingegeven waarde correct is.

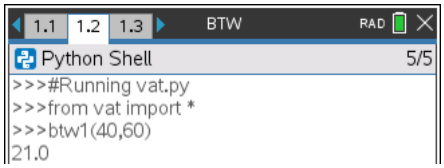
```
while True:
    getal=input("Getal n met 0 ≤ n ≤ 5: ")
    try:
        n=int(getal)
    except ValueError:
        print("VERKEERDE INPUT - Probeer opnieuw!")
        continue
    if n < 0 or n > 5:
        print("BUITEN DE GRENZEN - Probeer opnieuw!")
        continue
    break

print("De waarde van het getal n =",n)
```

Het error-type, ValueError, kan ook hier weggelaten worden.

d. *args

Veronderstel dat we de BTW willen berekenen op som van de netto-prijs van een aantal producten:

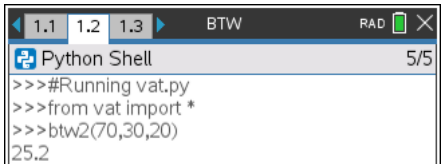
```
def btw1(a,b):
    return sum((a,b))*0.21
```



Wat als we de btw op meer dan twee producten willen berekenen?

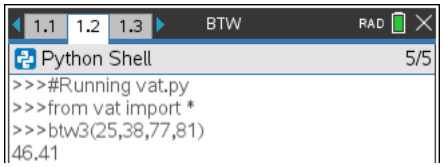
Een manier om dit te doen is het aantal argumenten te verhogen en een standaard waarde toe te kennen aan de argumenten.

```
def btw2(a=0,b=0,c=0,d=0,e=0):
    return sum((a,b,c,d,e))*0.21
```



Het starten van een parameter van een functie met een asteriks, *, maakt het mogelijk voor de functie om een willekeurig aantal argumenten te gebruiken. De functie beschouwt de argumenten als een tuple.

```
def btw3(*args):
    return sum(args)*0.21
```



e. Lambda

Lambda-uitdrukkingen zijn krachtige Python-tools die het toe laten om ad hoc anonieme functies te definiëren, zonder gebruik te maken van def. De structuur van lambda's is één enkele uitdrukking en niet een blok statements.

De onderstaande functie square() ziet er in lambda-vorm als volgt uit:

$f(x) = x^2$ $\qquad\qquad$ $x \longmapsto x^2$

```
def square(n):
    kwad=n**2          <------------>   lambda num : num**2
    return kwad
```

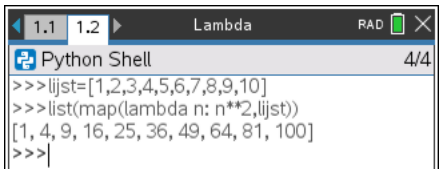
Normaal geef je aan een lambda-uitdrukking geen naam, maar toch even om een lambda-uitdrukking te demonstreren:

```
square = lambda num: num**2
```

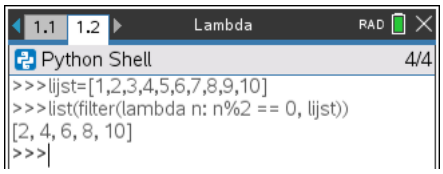
Hoe gebruik je een lamda-uitdrukking dan wel? Soms moet je een functie maar één keer uitvoeren in een programma en graag zonder een formele definitie van de functie.

Dan komt een lambda-uitdrukking goed van pas.

list(map(lambda n: n**2, lijst))



list(filter(lambda n: n%2 == 0, lijst))



f. Wat meer Palindroom-code ...

... gebruikmakend van string-functionaliteit

Code1

```
alfabet="abcdefghijklmnopqrstuvwxyz"

def strip_tekst(tekst):
    tekst = tekst.lower()
    str = ""
    for t in tekst:
        if t in alfabet:
            str = str+t
    return str

a=strip_tekst("Nelli plaatst op 'n parterretrap 'n pot staalpillen")
print(a)

def is_palindroom(tekst):
    l = len(tekst)
    for i in range(l/2):
        if tekst[i] != tekst[l-i-1]:
            return False
    return True

a=strip_tekst("Nelli plaatst op 'n parterretrap 'n pot staalpillen")
print(is_palindroom(a))
```

Code 2

```
def is_palindroom(tekst):
    a=list(tekst)
    b=a.copy()
    b.reverse()
    if b==a:
        return True
    else:
        return False

print(is_palindroom("meetsysteem"))
```

g. Programmeeropdrachten

Opdracht 1: Integer-input $n \geq 0$

Schrijf de code voor een input die enkel een geheel getal $n \geq 0$ aanvaardt en blijft vragen voor input totdat een aanvaardbare waarde is ingegeven:

- Indien de input geen geheel getal is, print "Verkeerde input – Probeer opnieuw",
- Indien de input een negatief geheel getal is, print "Negatief getal – Enter een positief getal".

Bereken m.b.v. van deze input-code en de onderstaande code $n!$ voor $n \geq 0$.

```
def fac(n):
    if n==0:
        return 1
    else:
        faculteit = 1
        for i in range(1,n+1):
            faculteit *= i
        return faculteit

# Input van enkel een geheel getal n ≥ 0.
......

print("De faculteit .....gem )
```

Opdracht 2: Dobbelen met Pascal


Blaise Pascal

De Franse Chevalier de Méré ontdekte bij het dobbelen dat het kansrijker was om in vier worpen met één dobbelsteen minstens een keer zes te gooien, dan in 24 worpen met twee dobbelstenen minstens een keer dubbel zes.

Daar hij dit helemaal niet verwachtte, vroeg hij uitleg aan Blaise Pascal (1623-1662).

Pascal vertelde hem dat de kans op winst respectievelijk 0,518 en 0,491 zijn.

De volgende codes simuleert beide dobbel-experimenten:

- worpen = [randint(1,6) for i in range(0,4)]
  6 in worpen
- worpen = [(randint(1,6),randint(1,6) for i in range(0,4)]
  (6,6) in worpen

Schrijf programma's die beide dobbel-experimenten uitvoert, met:

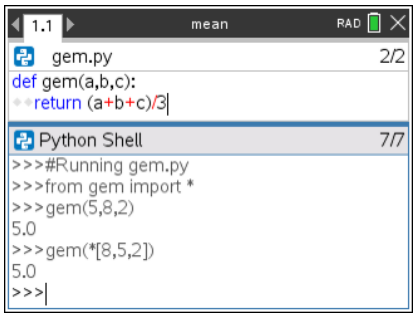
- na iedere uitvoering de optie om verder te stoppen (0) of verder te spelen (1),
- het aantal experimenten wordt bij gehouden, aantal,
- het aantal keren gewonnen wordt bijgehouden, winst.

Benader/simuleer de kans op winst voor beide experimenten.

Opdracht 3: Gemiddelde van getallen

Definieer een functie die van een willekeurig aantal getallen het gemiddelde berekent door b.v. gebruik te maken van *data als argument.

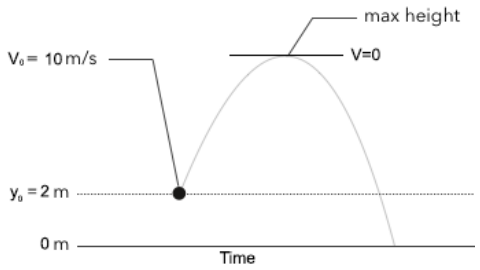
Omgekeerd kan een lijst (of tuple) voorafgegaan door een asteriks, *, gebruikt worden als de argumenten van een functie.



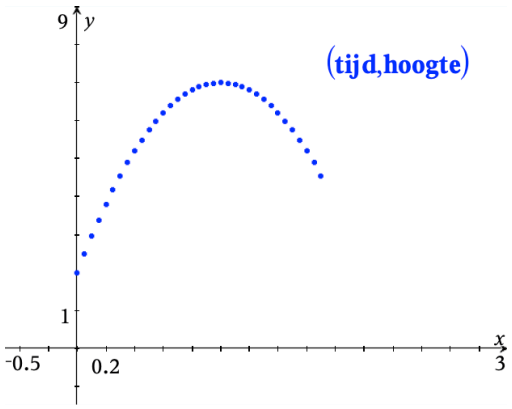
Opdracht 4: Gravitatie

Schrijf een programma om de maximale hoogte te vinden van een bal die recht omhoog wordt gegooid van af een hoogte van 2 meter met een beginsnelheid van 10 m/s.

$$y = \frac{1}{2}gt^2 + v_{y_0}y + y_0 \quad \text{met } g = -9{,}81\,\frac{m}{s^2}$$



- Stap 1
  Creëer d.m.v. lijstcomprehensie een lijst van tijdstippen, iedere 0,05 s.
- Stap 2
  Bereken voor de lijst uit Stap 1 de hoogte m.b.v. map() en lambda.
- Stap 3
  Bepaal het maximum van de berekende hoogtes met de max()-functie.
- Stap 4
  Bepaal het tijdstip behorende bij de maximale hoogte – lijst.index(element).
- Stap 5
  Exporteer de lijsten met de tijdstippen en hoogtes naar TI-Nspire-lijsten en teken hiermee een scatterplot in Graphs. Wanneer raakt de bal de grond?



20. Objectgeoriënteerd

Objectgeoriënteerd programmeren is een systeem gebaseerd op objecten die bestaan uit data/gegevens (attributen) en code (methodes) om deze gegevens te bewerken of verwerken.

In Python worden objecten gemodelleerd als instanties (elementen) van een klasse. Een klasse kan beschouwd worden als een blauwdruk die de definitie van een object bepaalt.

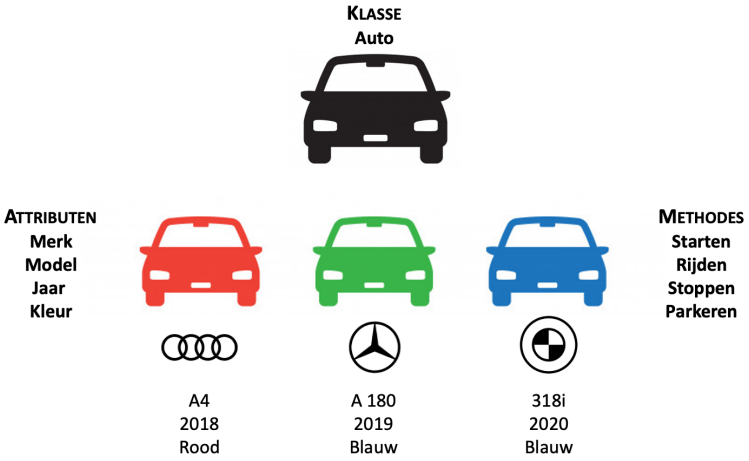
Een klasse bestaat uit:

- **Attributen** die de toestand/eigenschappen van een object vastleggen
- **Methodes** of functies die de toestand van het object aanpassen en het gedrag van een object bepalen.

Een speciale methode (de constructor) instantieert een object met zijn eigen waarden van de attributen; de toestand van een object.

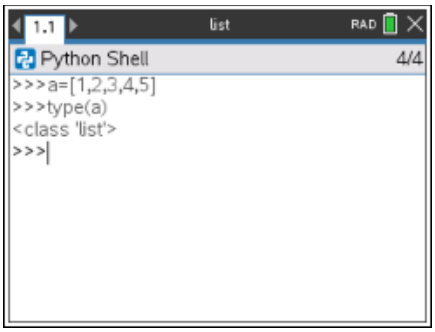
Object Oriented Programming (OOP) laat programmeurs toe hun eigen objecten en bijhorende methodes te construeren en zo hun code modulair op te bouwen.

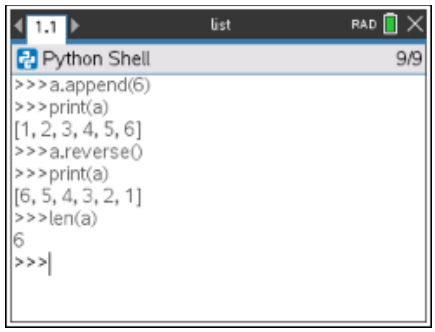
Klassen en objecten kan je vergelijken met objecten uit de ons omringende wereld zoals bv een auto.



In Python ***Everything is an object***.

In deze TI Python Bootcamp hebben we al veel gewerkt met objecten en methodes. Een voorbeeld hiervan is het data-type lijst met de methodes append() & reverse() en de speciale methodes print() & len().





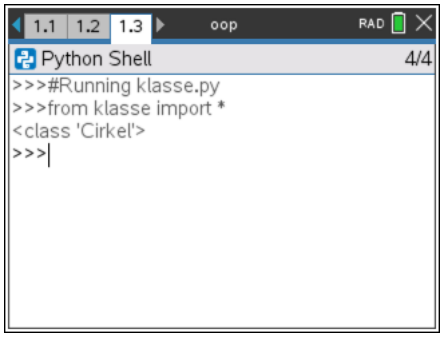
We bekijken de basics van objectgeoriënteerd programmeren in Python.

21. Klassen

User defined objecten kunnen gecodeerd worden met het keyword class.

We illustreren de syntax van het coderen van klassen met het concept cirkel.

```
# Definitie nieuw object type
class Cirkel:
    pass

# Declaratie object van de klasse Cirkel
c=Cirkel()

print(type(x))
```



Het statement pass kan je gebruiken als een placeholder voor code. Wanneer pass wordt uitgevoerd, gebeurt er niets maar het geeft geen error. Lege code-blokken genereren namelijk een error.

Een Python-conventie geeft aan dat een klasse-naam start met een hoofdletter; net zoals we klein letters gebruiken voor variabelen.

22. Attributen

Een cirkel is analytisch volledig bepaald door zijn middelpunt en de straal. Het middelpunt en de straal gaan we coderen als de attributen(eigenschappen) van een object van de klasse Cirkel.

De syntax voor het creëren van een attribuut is b.v. self.rad = rad als onderdeel van de speciale methode __init__().

De methode __init__() initialiseert de attributen van een object en deze methode wordt steeds uitgevoerd bij het aanmaken van een object.

De algemene syntax voor het coderen van de attributen van een klasse is de volgende:

```
class Cirkel:
    def __init__(self,parameter1,parameter2):
        self.parameter1 = parameter1
        self.parameter = parameter2
```

Het keyword **self** representeerd de instantie van de klasse en wordt gebruikt om naar zichzelf te wijzen. Het is niet noodzakelijk de naam self te gebruiken maar het is een afspraak tussen Python-programmeurs.

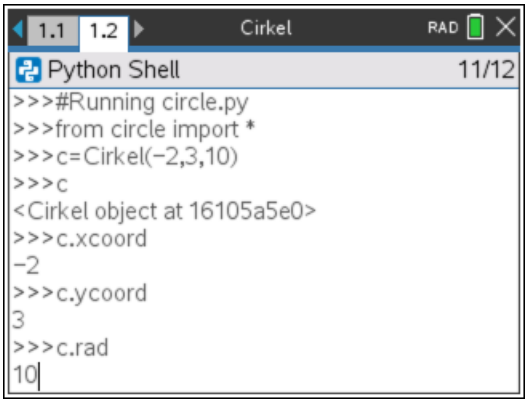
Deze syntax kan wat eigenaardig overkomen (b.v. 3x parameter1). Ook dit is een python-afspraak om telkens driemaal hetzelfde woord te gebruiken. Het is niet noodzakelijk en het zal geen error geven indien niet zo.

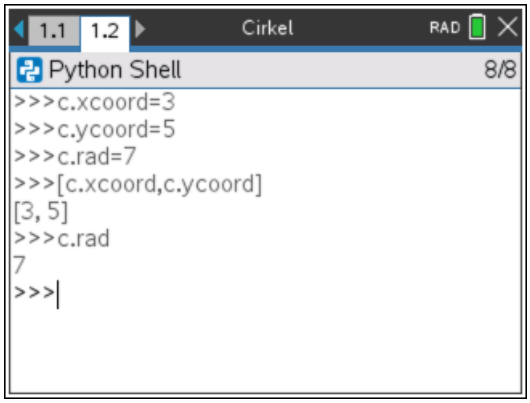
Hieronder twee dezelfde klassen voor het object Cirkel:

```
class Cirkel:
    def __init__(self,xcoord,ycoord,rad):
        self.xcoord = xcoord
        self.ycoord = ycoord
        self.rad = rad
```

```
class Cirkel:
    def __init__(self,xc,yc,straal):
        self.xcoord = xc
        self.ycoord = yc
        self.rad = straal
```

Na het declareren van een cirkel object – c = Cirkel(-3,10,2) wordt een object Cirkel aangemaakt in het geheugen. De attributen(eigenschappen) kunnen als volgt opgeroepen worden en eventueel aangepast.

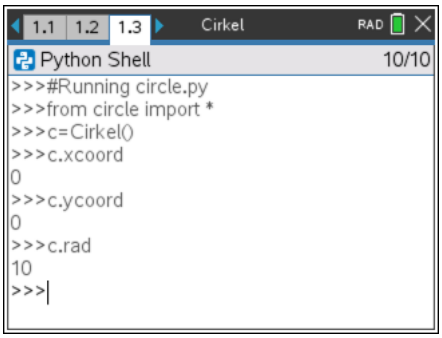




Voor de argumenten van __init__ (alsook voor andere methodes) kan als volgt aan de argumenten een standaard-waarde toegekend worden:

```
def __init__(self,xcoord=0,ycoord=0,rad=10).
```

Een Cirkel-object kan in dit geval gedeclareerd worden zonder argumenten c = Cirkel().

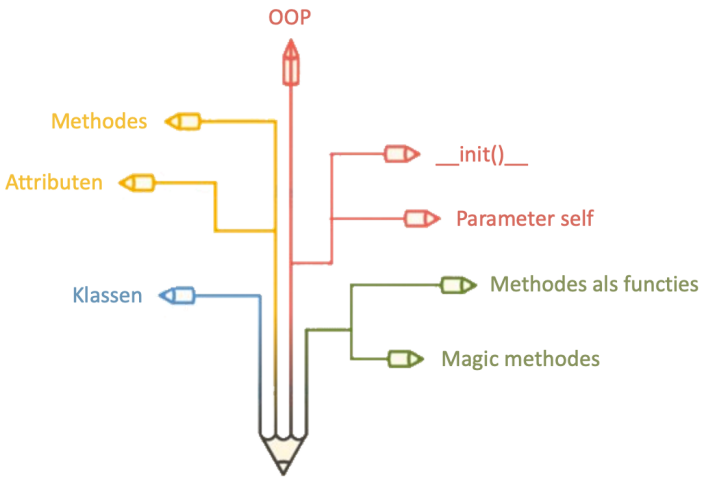


Tot hiertoe hebben we de volgende terminologie van object georiënteerd programmeren geïllustreerd:

- een **klasse**, het model of de standaard van de mogelijkheden wat een object kan zijn en doen
- een **object**, de instantie van een klasse m.a.w. een concrete entiteit van een klasse
- een **instantie**, is als een object maar laten we even terug naar het concept auto:
  - een blauwdruk voor een auto-ontwerp is de klasse-beschrijving
  - alle auto's die op basis van die blauwdruk zijn vervaardigd, zijn objecten van die klasse
  - uw auto die van die blauwdruk is gemaakt, is een instantie van die klasse.

De termen instantie en object worden vaak door elkaar gebruikt en de meest voorkomende term is object.

In het volgende deel bespreken we **methodes** van een klasse.



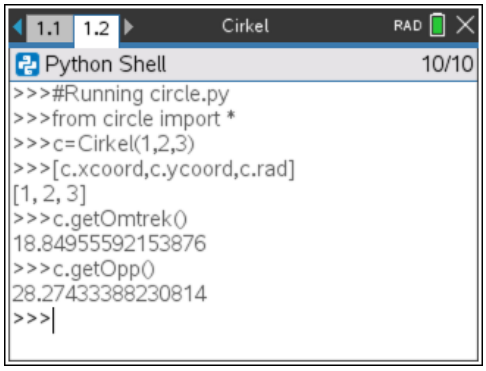
23. Methodes

Methodes zijn functies gedefinieerd in een klasse. Methodes worden gebruikt om operaties uit te voeren op/met de attributen van een klasse.

Een methode is een functie op een object van een klasse die het object zelf aanspreekt met het klasse-argument self.

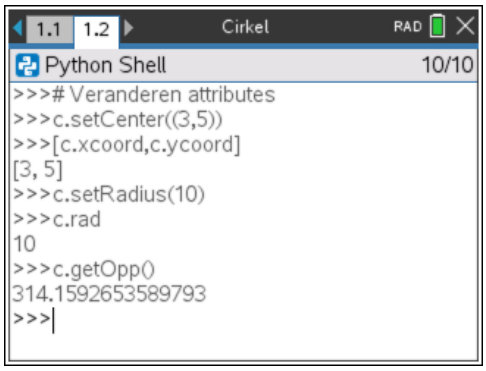
We illustreren enkele methodes voor de klasse Cirkel voor het berekenen van de omtrek en de oppervlakte.

```
from math import *
from ti_draw import *

class Cirkel:

    def __init__(self,xcoord,ycoord,rad):
        self.xcoord = xcoord
        self.ycoord = ycoord
        self.rad = rad

    def getOmtrek(self):
        return 2*pi*self.rad

    def getOpp(self):
        return pi*self.rad**2
```



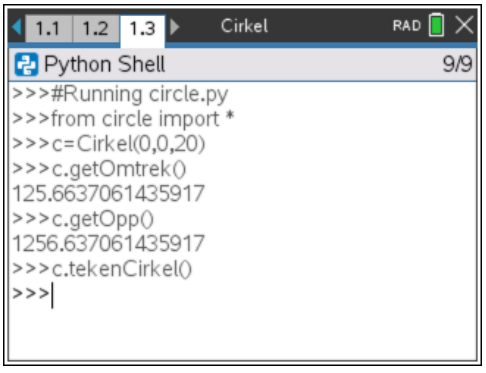
Methodes kunnen ook gebruikt worden om de attributen van een object te veranderen

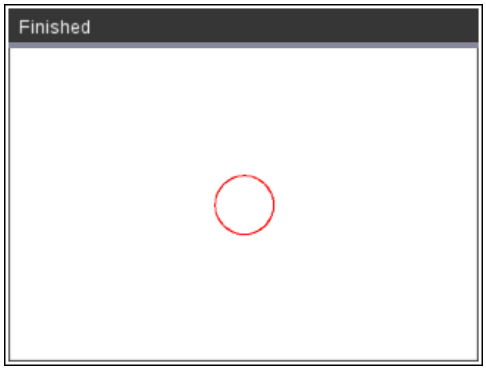
```
    def setRadius(self,newrad):
        self.rad=newrad

    def setCenter(self,newcenter):
        self.xcoord,self.ycoord=newcenter
```



en een object grafisch voor te stellen

```
    def tekenCirkel(self):
        w,h=get_screen_dim()
        set_window(-w/2,w/2,-h/2,h/2)
        set_color(255,0,0)
        draw_circle(self.xcoord,self.ycoord,self.rad)
```





24. Magic methodes

Voor Python-klassen kunnen speciale methodes, magic methodes, geïmplementeerd worden, gebruikmakend van speciale syntax. We maakten al kennis met __init__(), de methode die de attributen van de klasse definieert.

We bekijken nog twee andere magic methodes:

- __str__() string-representatie van een object
- __len__() geeft de (user defined) lengte van een object

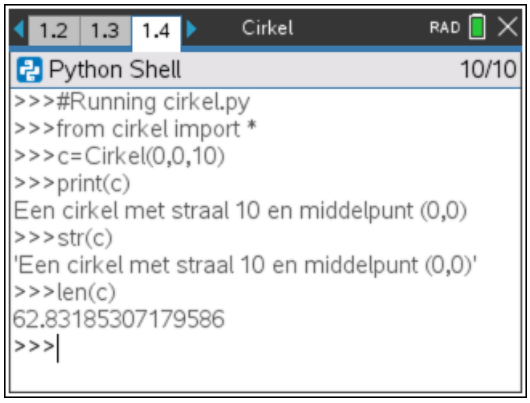
We herschrijven de klasse Cirkel als volgt:

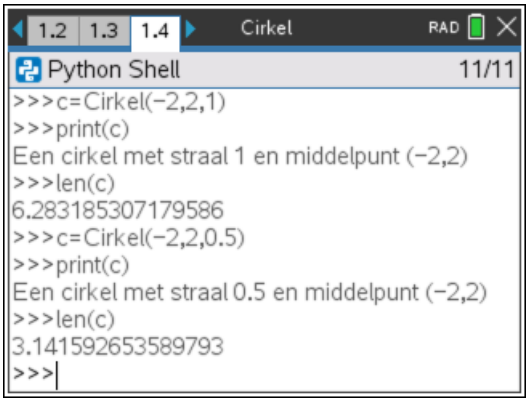
```
from math import *
class Cirkel:
    def __init__(self,xcoord,ycoord,rad):
        self.xcoord=xcoord
        self.ycoord=ycoord
        self.rad=rad
        self.omtrek=2*pi*rad
        self.opp=pi*rad**2
```

En we definiëren de volgende __str__() en __len__ magic methodes voor de klasse Cirkel:

```
    def __str__(self):
        return "Een cirkel met straal {} en middelpunt ({},{})".format(self.rad,self.xcoord,self.ycoord)

    def __len__(self):
        return self.omtrek
```

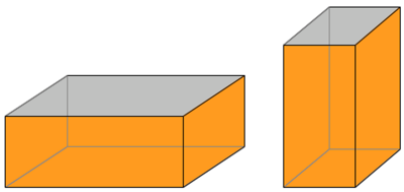
Deze methodes voeren we als volgt uit op een object:





Programmeeropdrachten

Opdracht 1: Een balk



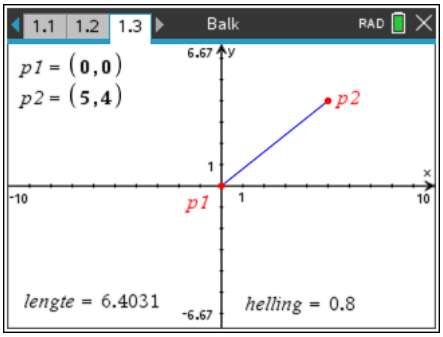
Definieer een klasse Balk() met
- als attributen de lengte, breedte en hoogte en
- als methodes het volume en de oppervlakte

Opdracht 2: Lengte en helling van een segment

a. Definieer een klasse Segment() met
   - als attributen de coördinaten van begin- en eindpunt als tuples en
   - als methodes de lengte en de helling van het segment

b. Codeer een methode die de coördinaten van het begin- en eindpunt van een segment uitvoeren naar TI-Nspire CX-variabelen waarmee het segment in Graphs getekend wordt.

c. Codeer een methode die de coördinaten van het begin- en eindpunt van een segment in de TI-Nspire CX-applicatie Graphs naar een object van de klasse Segment() importeren.

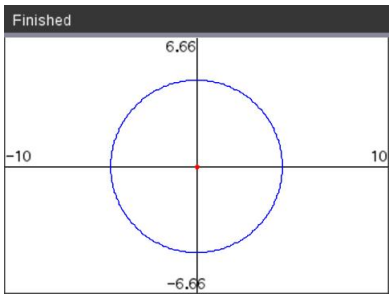


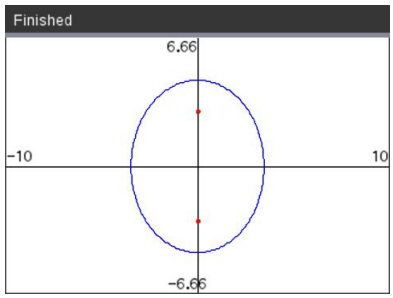
Opdracht 3: Kegelsnede

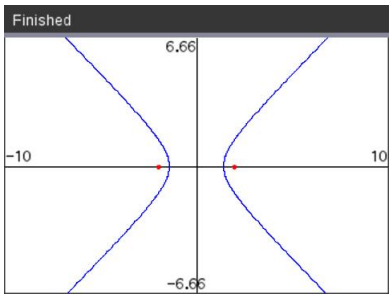
Programmeer een klasse Kegelsnede() met als attributen/argumenten a, b en $c > 0$ met a, b, c parameters van de vergelijking $\frac{x^2}{a} + \frac{y^2}{b} = c$.

Deze vergelijking stelt een cirkel, ellips of hyperbool voor afhankelijk van de waarde van a, b, c :
- $a = b > 0$ ⇒ de vergelijking bepaalt een cirkel
- $a, b > 0$ en $a \neq b$ ⇒ de vergelijking bepaalt een ellips
- $a \cdot b < 0$ ⇒ de vergelijking bepaalt een hyperbool
- anders ⇒ de vergelijking bepaalt geen kegelsnede

Definieer methodes die op basis van de argumenten a, b en c de kegelsnede bepaalt, onderstaande karakteristieken genereert en de kegelsnede plot in een orthonormaal assenstelsel.







Verdieping

a. Enkele karakteristieken van OOP

i. Inheritance (overerving)

Overerving is een manier om nieuwe klassen te definiëren op basis van bestaande klassne. Een nieuwe child-klasse die overerft van een bestaande parent-klasse, neemt bepaalde attributen en methodes over van de parent-klasse. M.a.w. de child-klasse gebruikt code van de parent-klasse maar kan ook nieuwe attributen en methodes toevoegen. Terminologie die hier ook wel gebruikt wordt is basis-klasse en afgeleide (sub)klasse.

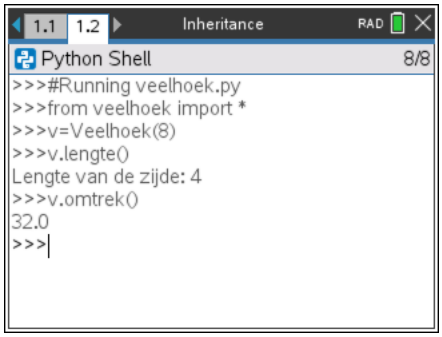
De python-syntax is de volgende:

```
class BasisKlasse():
    Blok

class AfgeleideKlasse(BasisKlasse):
    Blok
```

Voorbeeld
Omtrek van een regelmatige veelhoek

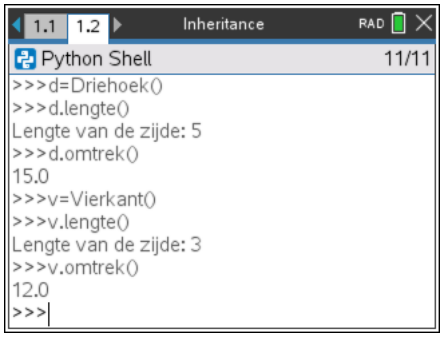
We definiëren als basis-klasse de klasse Veelhoek() met als attributen het aantal zijden en als methodes input van de lengte van de zijde en het berekenen van de omtrek.

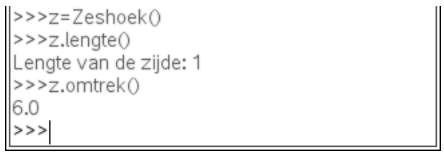
```
class Veelhoek():
    def __init__(self, aantal_zijden):
        self.aantal_zijden = aantal_zijden
    def lengte(self):
        self.lengte = float(input("Lengte van de zijde: "))
    def omtrek(self):
        return self.aantal_zijden*self.lengte
```



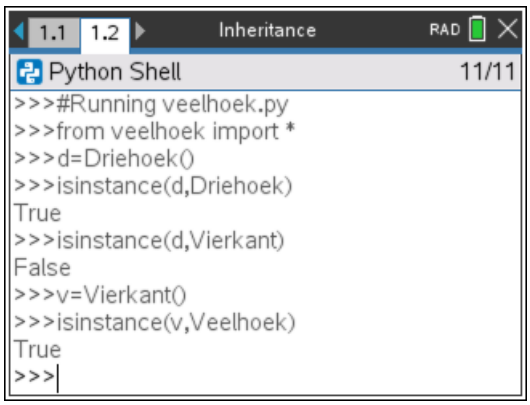
Als afgeleide klassen definiëren we de subklassen Driehoek(), Vierkant(), Vijfhoek() en Zeshoek(). Al deze klassen kunnen gebruik maken van de methodes van de klasse Veelhoek().

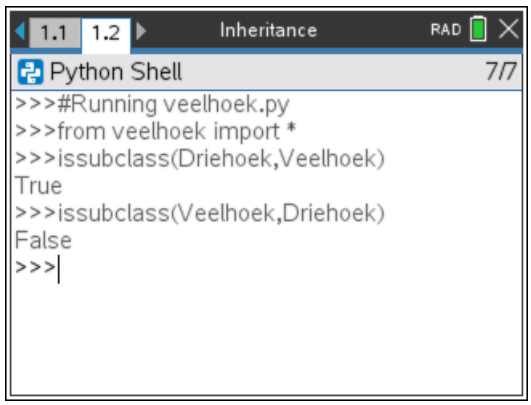
```
class Driehoek(Veelhoek):
    def __init__(self):
        Veelhoek.__init__(self,3)
class Vierkant(Veelhoek):
    def __init__(self):
        Veelhoek.__init__(self,4)
class Vijfhoek(Veelhoek):
    def __init__(self):
        Veelhoek.__init__(self,5)
class Zeshoek(Veelhoek):
  def __init__(self):
    Veelhoek.__init__(self,6)
```





Met het statement isinstance() kan je checken of een object een instantie van een klasse en met issubclass() controleer je klasse-inheritance.

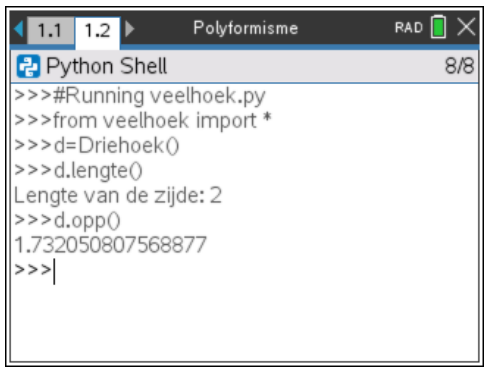


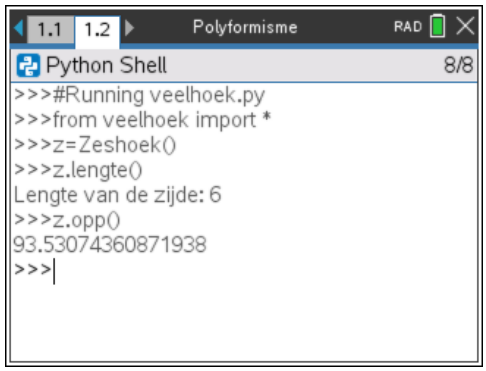


ii. Polymorfisme

Voor de subklassen van Veelhoek() definiëren we de methode opp() voor de oppervlakte van de veelhoek.

```
from math import *
class Driehoek(Veelhoek):
    def __init__(self):
        Veelhoek.__init__(self,3)

    def opp(self):
        return sqrt(3)/4*self.lengte**2

class Vierkant(Veelhoek):
    def __init__(self):
        Veelhoek.__init__(self,4)

    def opp(self):
        return lengte**2

class Vijfhoek(Veelhoek):
    def __init__(self):
        Veelhoek.__init__(self,5)

    def opp(self):
        return (5*self.lengte**2)/(4*sqrt(5-2*sqrt(5)))

class Zeshoek(Veelhoek):
    def __init__(self):
        Veelhoek.__init__(self,6)

    def opp(self):
        return 3*sqrt(3)*self.lengte**2/2
```





Voor iedere klasse definiëren we dezelfde methode met telkens een andere implementatie. Het uitvoeren van de methode runt telkens een andere code. Dit noemen we polymorfisme (veelvormigheid).

De algemene formule $self.opp = \frac{self.aantal\_zijden \cdot self.lengte}{4\tan\left(\frac{\pi}{self.aantal\_zijden}\right)}$ voor de klasse was ook een optie.

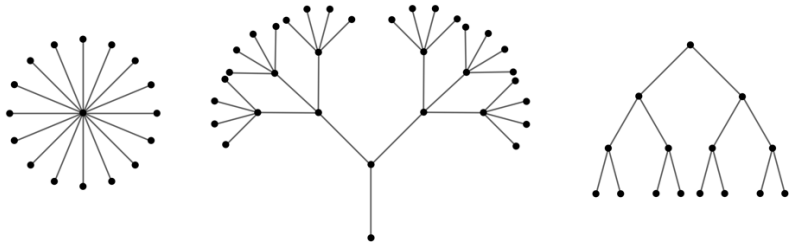
b. Prim-algoritme

Het Prim-algoritme is een algoritme om de minimale opspannende boom van een graaf te bepalen.

Even wat grafen-terminologie.

Een acyclische graaf een graaf is zonder cykel (= pad met lengte groter dan nul van een punt naar zichzelf); ook wel een bos genoemd. Een boom is samenhangende acyclische graaf.

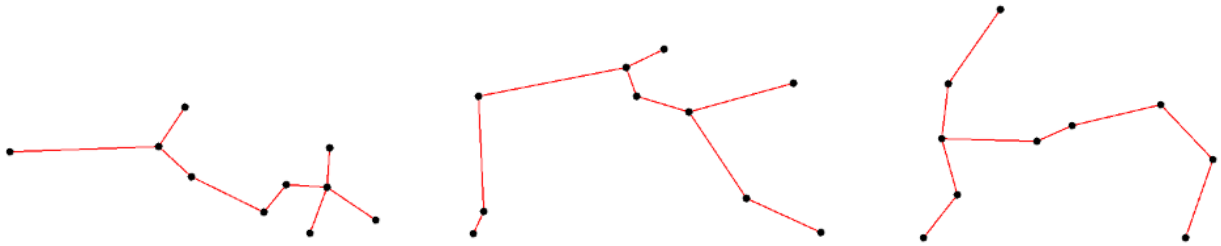
Hieronder wat bomen die samen een bos vormen:



Een opspannende boom is een deelgraaf die alle punten van de graaf bevat. Indien we aan de punten een gewicht toekennen, bv de kost of de afstand van een wandeling tussen twee punten, spreken we een gewogen boom.

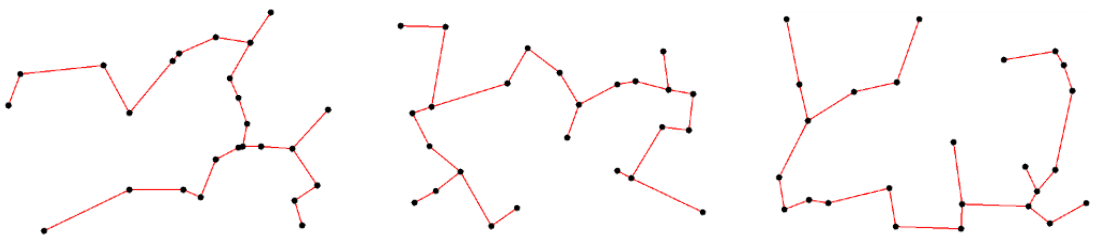
Het Prim-algoritme bepaalt de minimale (gewogen) opspannende boom, m.a.w. de boom met minimale gewicht, kost of afstand. We coderen een Prim-algoritme met als gewicht de Euclidische afstand tussen de punten.

Voor een graaf tekenen we de Euclidisch minimale opspannende boom (EMST – Euclidean minimum spanning tree). Een EMST verbindt een verzameling punten met lijnen zodat de totale lengte van de lijnen minimaal is en zodat ieder punt kan bereikt worden vanuit ieder ander punt door deze lijnen te volgen.



We bouwen het algoritme als volgt op voor een verzameling van punten:

1. Start een boom met een willekeurig punt p (verbonden) en een lijst met punten die nog niet toegevoegd zijn aan de boom (niet verbonden). Bij de start zijn dit alle punten uitgezonderd het willekeurig gekozen punt p.
2. Zoek voor dit punt p het punt q uit de lijst van niet verbonden punten waarvoor geldt dat de afstand minimaal is.
    a. Verbind de punten p en q,
    b. verwijder q van de lijst van niet verbonden punten en
    c. voeg q toe aan de verbonden punten van de boom.
3. Herhaal stap 2 voor alle verbonden punten tot de er geen niet verbonden punten meer zijn



De onderstaande code maakt gebruikt van de TI Python-module TI-Draw; specifiek voor de TI grafische technologie.

We starten met het definiëren van objecten, methodes en functies:

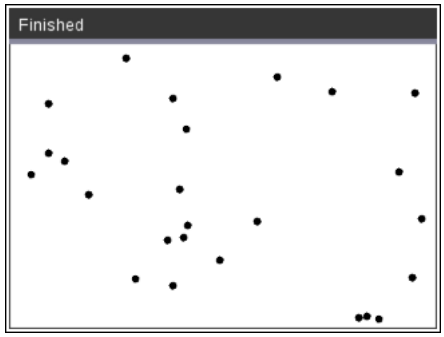
```
from ti_draw import *
from random import *
from math import *

class Punt():
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def teken(self):
        set_color(0,0,0)
        fill_circle(self.x,self.y,r)

def afstand(p,q):
    return sqrt((p.x-q.x)**2+(p.y-q.y)**2)

def lijn(p,q):
    set_color(255,0,0)
    draw_line(p.x,p.y,q.x,q.y)

def teken(punten):
    for p in punten:
        p.teken()

def emstBoom(punten):
    verbonden=[ ]
    niet_verbonden=[ ]
    for p in punten:
        niet_verbonden.append(p)
    rand = randint(0,len(punten)-1)
    p = punten[rand]
    verbonden.append(p)
    niet_verbonden.remove(p)
    while niet_verbonden !=[ ]:
        a=500
        for p in verbonden:
            for q in niet_verbonden:
                d=afstand(p,q)
                if d<a:
                    a=d
                    p1=p
                    q1=q
        lijn(p1,q1)
        verbonden.append(q1)
        niet_verbonden.remove(q1)
```

We bepalen het scherm en generen de punten,

```
w,h = get_screen_dim()
r,n = (3,25)

punten = [ ]
for i in range(n):
    x = randint(r,w-r)
    y = randint(r,h-r)
    punten.append(Punt(x,y))
```

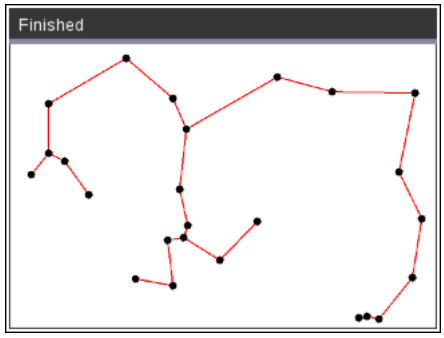
tekenen de punten en

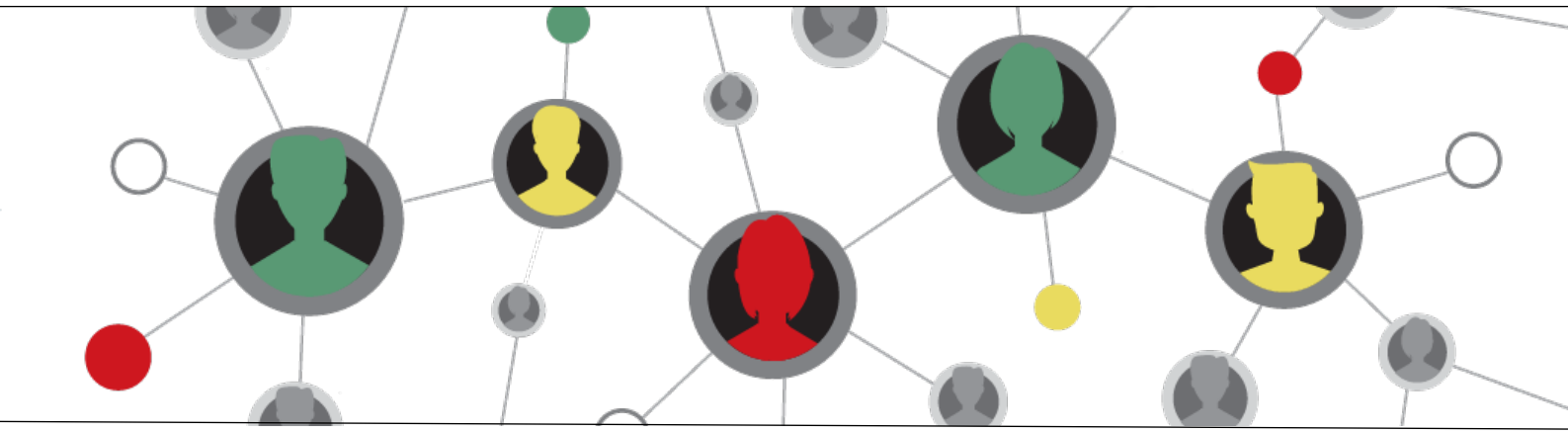
```
w,h = get_screen_dim()
r,n = (3,25)

punten = [ ]
for i in range(n):
    x = randint(r,w-r)
    y = randint(r,h-r)
    punten.append(Punt(x,y))

teken(punten)
```



bepalen de EMST-boom,

```
emstBoom(punten)
```





T³ NEDERLAND T³ VLAANDEREN

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

Texas Instruments