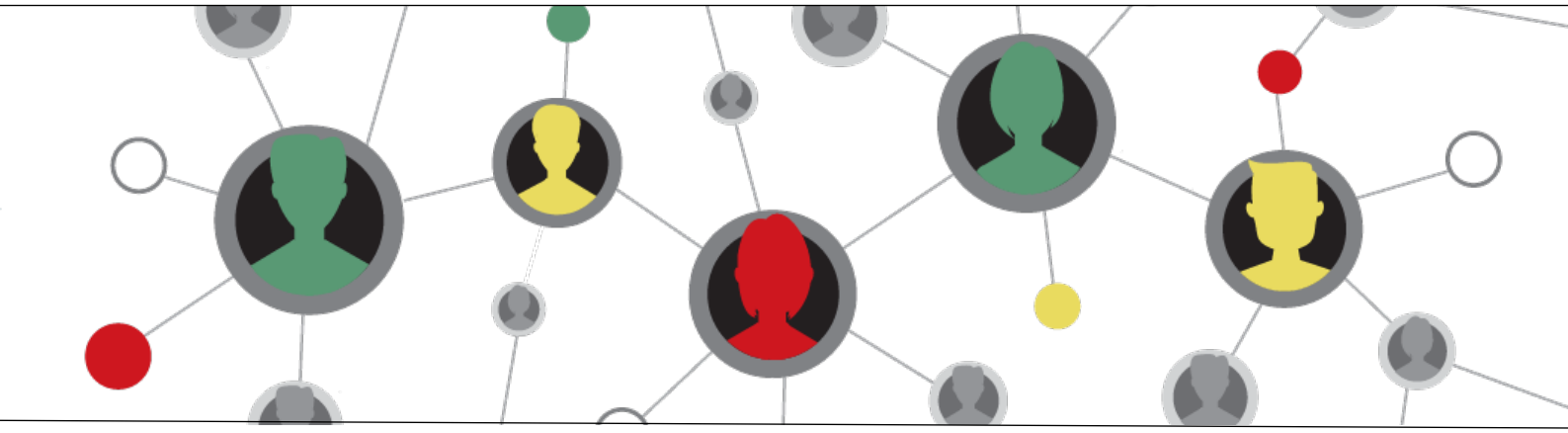
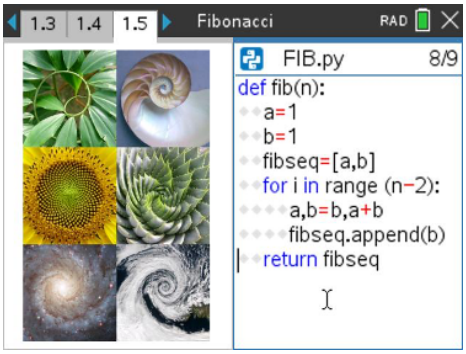
TI Python BootCamp

Deel 2 – Iteraties & Functies





T³ NEDERLAND

Teachers Teaching with Technology™

T³ VLAANDEREN

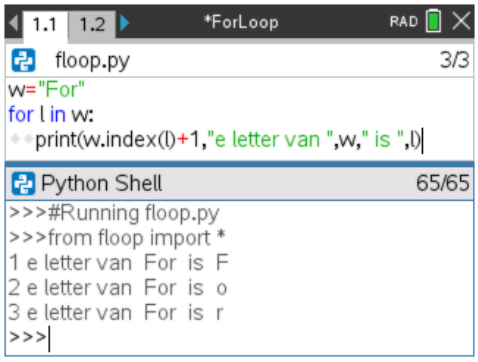
1. For-lus

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:

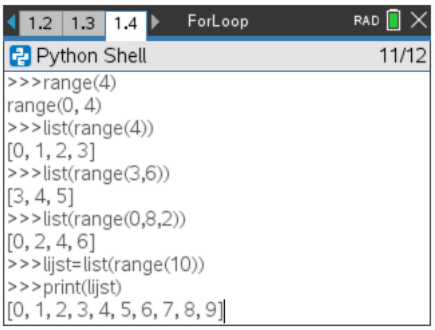
```
for index in object:
    block
```

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: $P(zes) = \frac{1}{6}$.

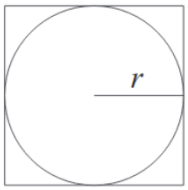
```
from random import *

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.

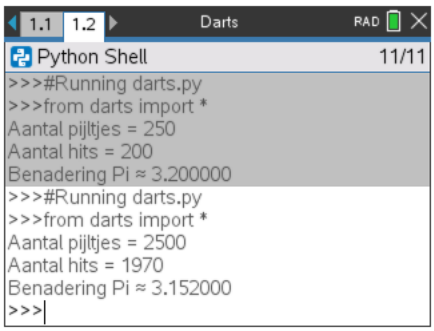


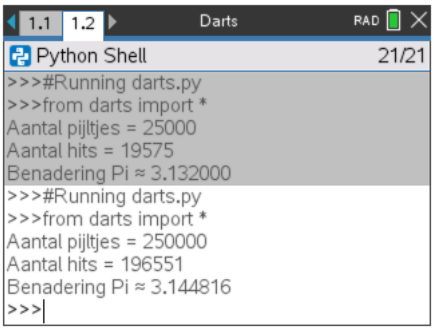
De verhouding van het aantal punten in de cirkel tot het totale aantal gegenereerde punten geeft als volgt een benadering voor π:

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

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))
```





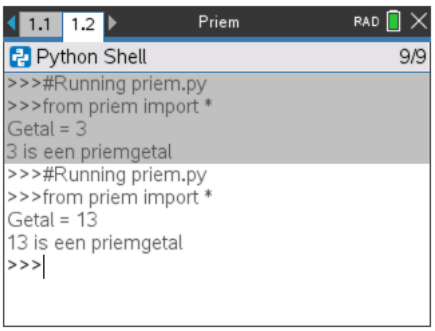
Voorbeeld 3

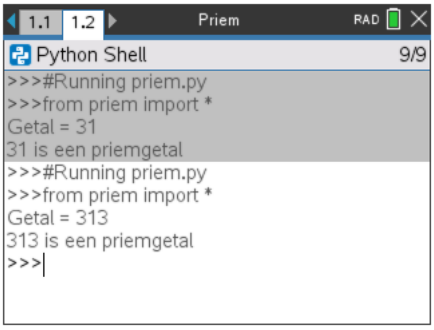
Wanneer is een getal een priemgetal?

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")
```





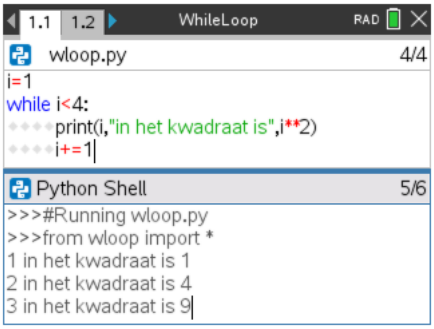
2. While-lus

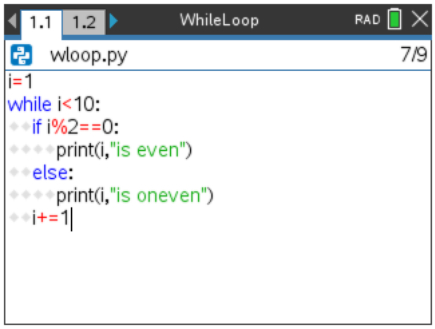
Een while-lus wordt gebruikt voor het uitvoeren van code of blokken van code zolang aan een bepaalde voorwaarde voldaan is.

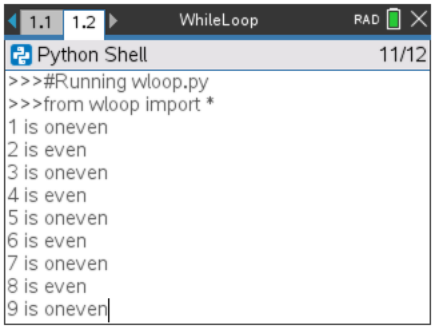
De structuur van een while-lus is als volgt:

```
while BooleanExpr:
    block
```

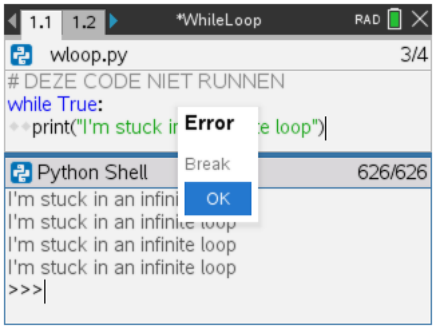
Twee eenvoudige voorbeelden voor het illustreren van de syntax van een while-lus:







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

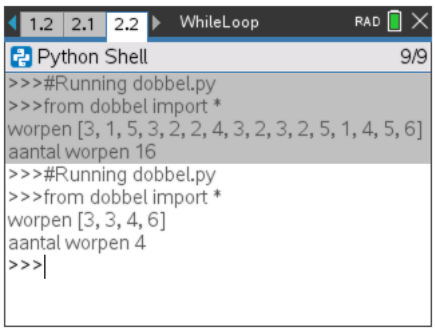
Afhankelijk van het programma, wordt het programma niet altijd onmiddellijk onderbroken; best dan de knop langer ingedrukt te houden

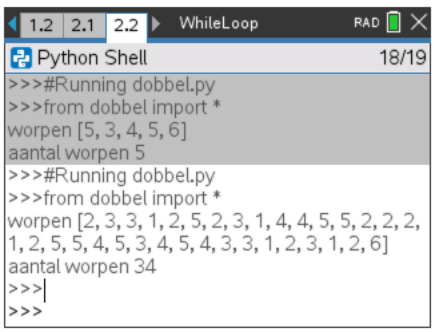
Voorbeeld 1

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.

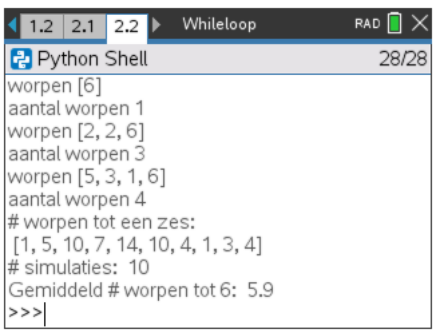
```
from random import *

aantal=0
ogen=0
worpen=[ ]

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

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

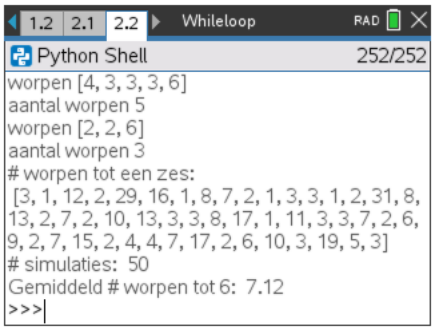


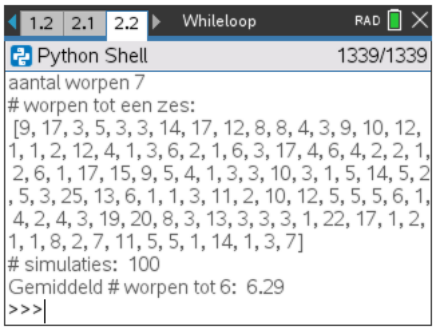


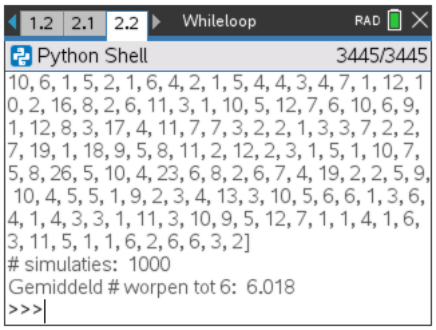
We voegen code toe (for-lus) om deze simulatie een aantal keren na mekaar uit te voeren en telkens het aantal worpen tot zes ogen op te slaan in een lijst. We eindigen het programma met het berekenen van het gemiddelde van deze lijst.

```
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))
```



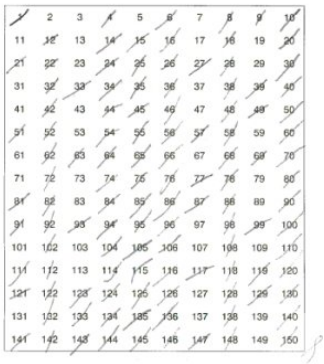






Voorbeeld 2

Zeef van Eratosthenes



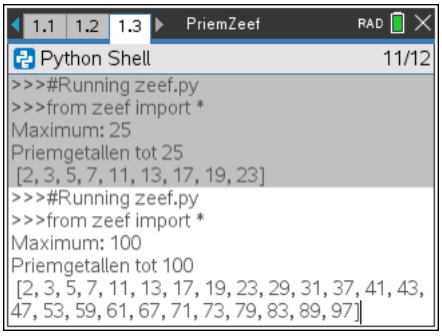
Dit zeer lang gekende algoritme voor het vinden van priemgetallen werkt als volgt:

- 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)
```

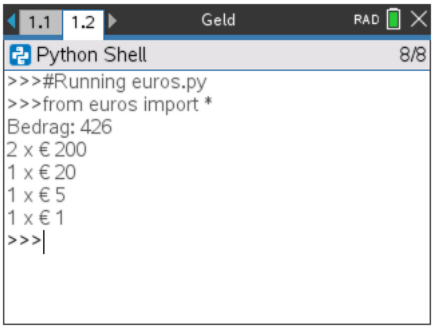


Voorbeeld 3

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.

2.1. 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")
```

2.2. 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.

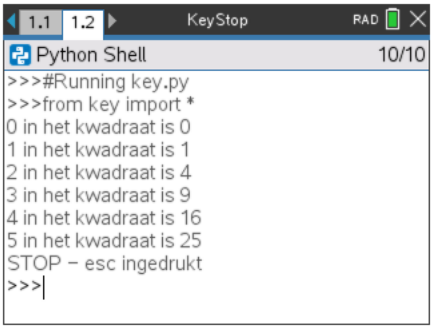

```
key=0
while key != "esc":
    block
    key=get_key()
```

We gebruiken het sleep()-statement van de time-module om het uitvoeren van de code te pauzeren.

De output van kwadraten blijft op het scherm – per 1 seconde – verschijnen tot het indrukken van de esc-toets.

Merk op dat meerdere code op 1 regel kunt plaatsen door met ; de code te scheiden.

```
from ti_system import *
from time import *

i=0 ; key=0

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

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



De output van kwadraten blijft op het scherm – per 1 seconde – verschijnen tot het indrukken van de esc-toets.

Merk op dat meerdere codes met ; op eenzelfde regel kunnen geplaatst worden.

1. 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:

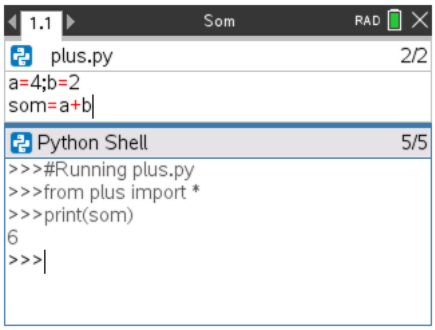
```
def function(argument):
    block
```

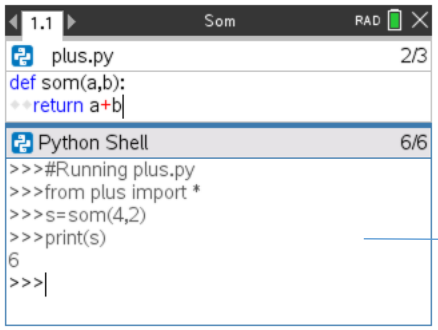
We illustreren de syntax met het voorbeeld de som van twee getallen. Het return-statement laat toe een functie een resultaat te returneren, een resulaat dat b.v. bewaard kan worden als een variabele.

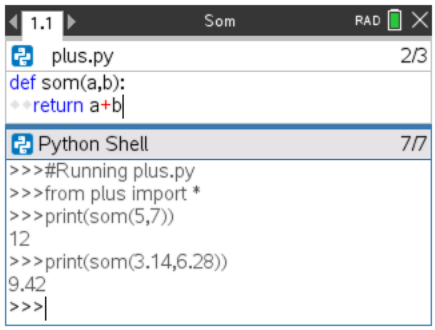
```
a=4
b=2
som=a+b
```

→

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



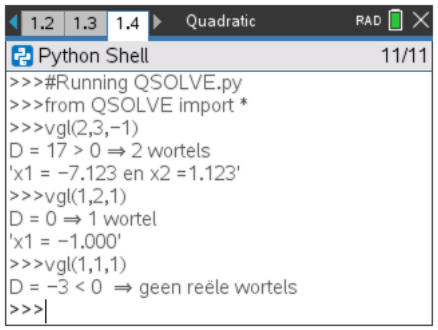


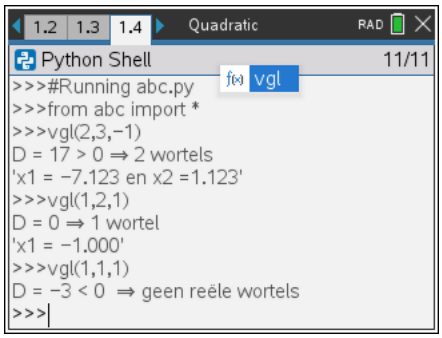


Voorbeeld 1 – De abc-formule

Het oplossen van een verglijking van de tweede graad $ax^2 + bx + c = 0$ kan als volgt geprogrammeerd 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")
```





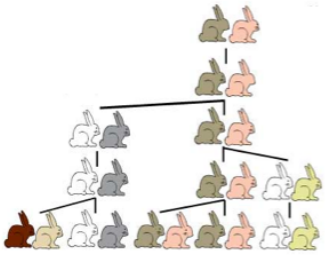
In plaats van het intikken van een gedefinieerde functie, kan de functie ook opgeroepen worden met behulp van de var-knop.

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?

Indien 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.

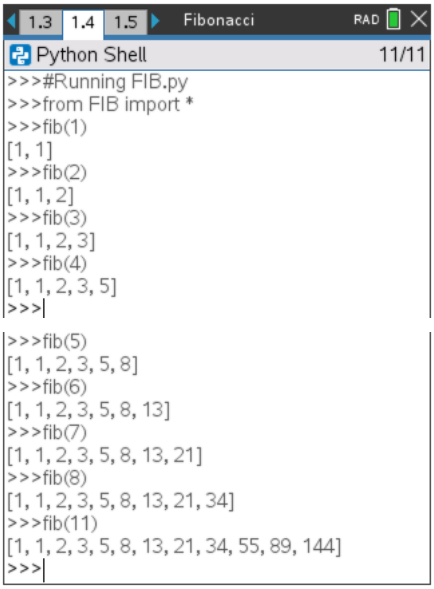
De rij van Fibonacci wordt als volgt op gebouwd:

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

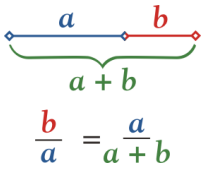
Men kan bewijzen dat: $F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$.

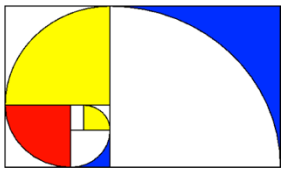
Het volgende programma berekent de rij van Fibonacci:

```
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.





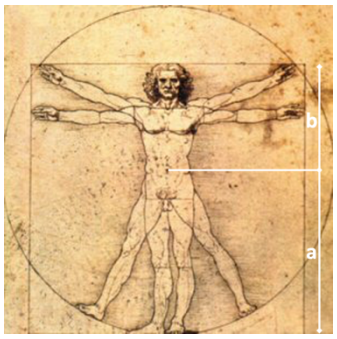
Deze verhouding wordt ook wel het gouden getal genoemd:

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

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:

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

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

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

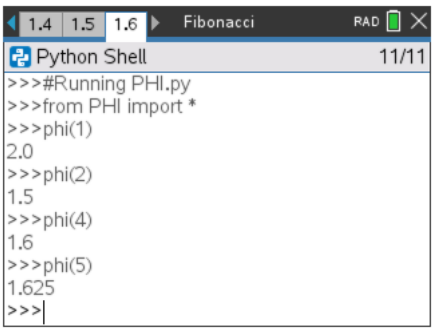
$$\varphi \approx 1 + \frac{1}{1+\frac{1}{1+\frac{1}{1+1}}} = \frac{8}{5} = 1.6$$

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

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

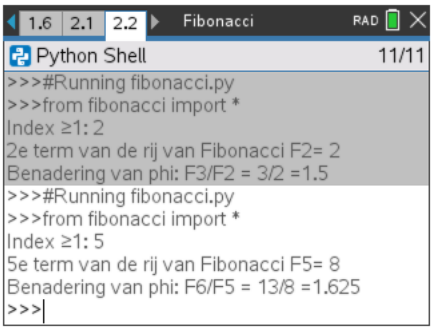
Voor het benaderen van φ door de verhouding van twee opeenvolgende Fibonacci getallen gebruiken we zojuist gedefinieerde functie fib() uit het FIB.py programma. We starten de code met from FIB import *.

```
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)))
```

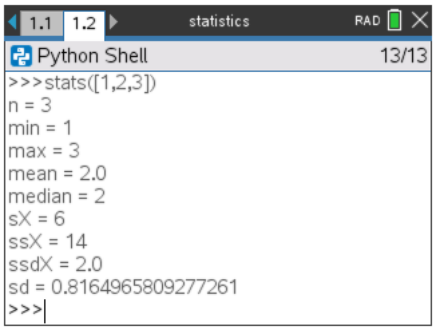


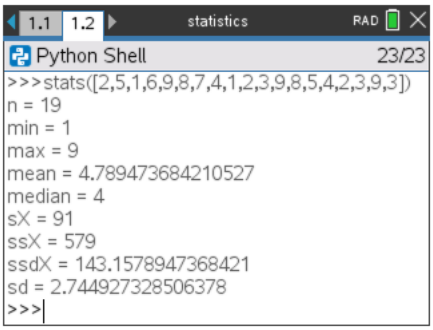
Voorbeeld 3 – Statistische kengetallen

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
```

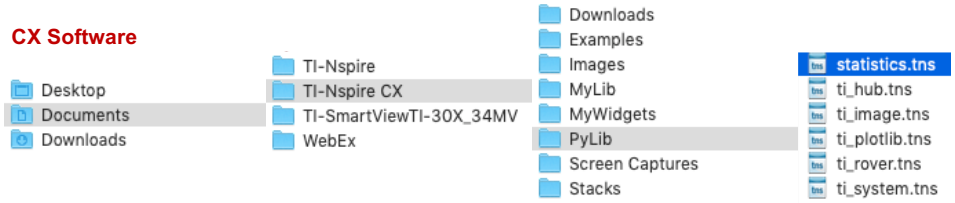
```
def median(list):
    slist=sorted(list)
    if len(list)%2:
        mid=int(len(list)/2)+1
        return slist[mid-1]
    else:
        mid=int(len(list)/2)
        return (slist[mid]+slist[mid-1])/2

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
```



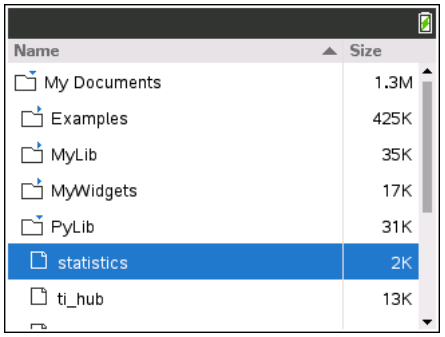


Om deze functies te gebruiken als een zelf gedefinieerde module, plaats je het tns-document met het programma met de functies in de PyLib-folder (... > Documents > TI-Nspire CX > PyLib).

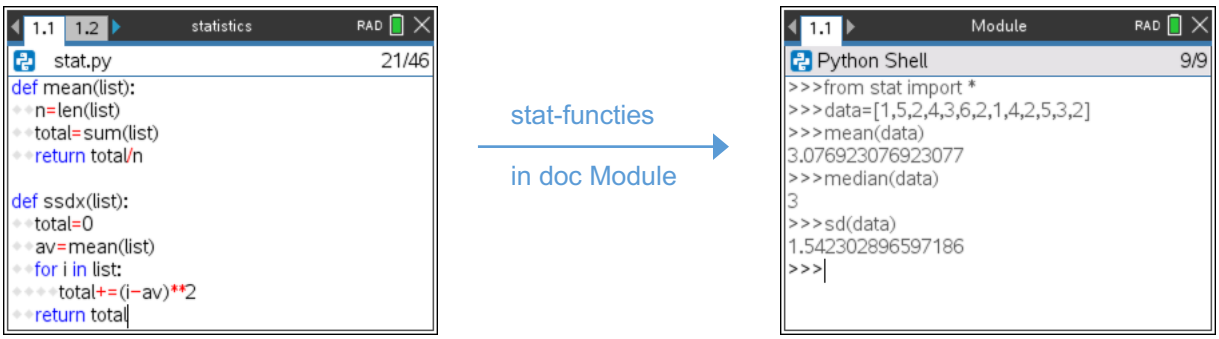
CX Software



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:



2. 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.

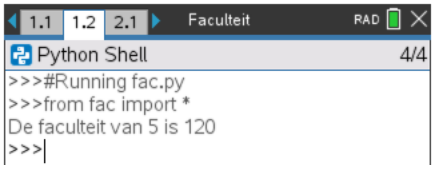
Met behulp van recursieve functies kan het probleem worden onderverdeeld in sub-problemen die eenvoudig kunnen worden opgelost. Maar het is soms moeilijker om de logica van recursieve functie te volgen.

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: $n! = n \cdot (n-1)!$

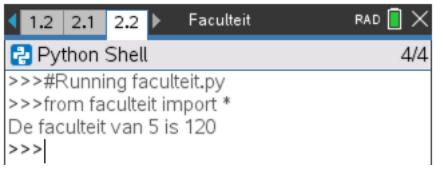
Met de definitie

```
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))
```



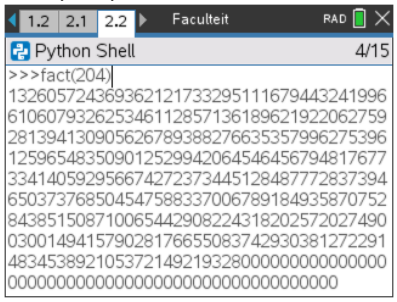
Met recursie

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

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

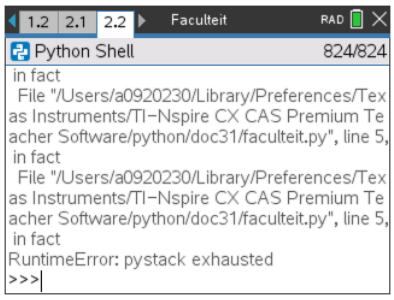


Python legt recursie een beperking op omdat elke keer een functie zichzelf aanroept, tussenliggende waarden bewaard moeten worden in het geheugen. Voor het bovenstaande programma kan hoogstens 204! berekend worden.

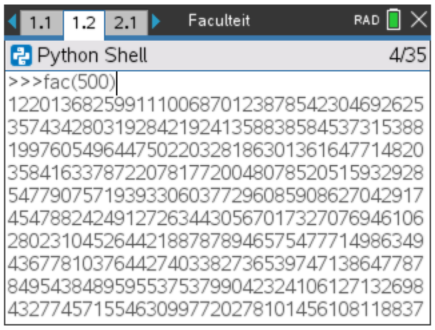
fact(204)

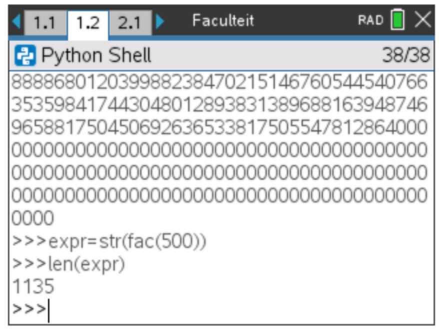


fact(205)



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.





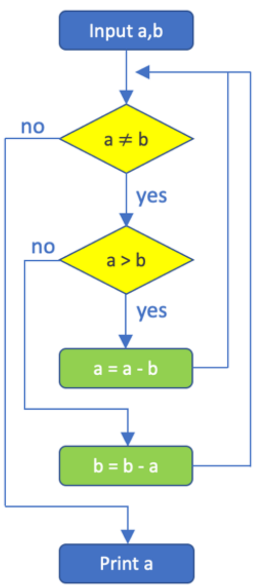
Voorbeeld 2 – GGD

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

Algoritme van Euclides

Zolang als 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

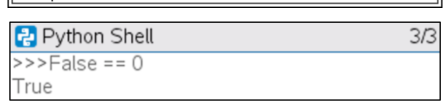



```
a=int(input("a= "))
b=int(input("b= "))
x,y = a,b
while x!=y:
♦♦if x>y:
♦♦♦♦x=x-y
♦♦else:
♦♦♦♦y=y-x
print("De grootste gemene deler van {} en {} = {}".format(a,b,x))
```

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: .



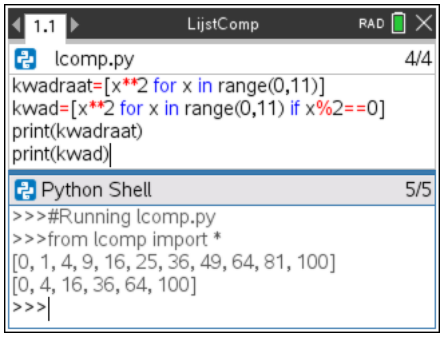
GGD recursief geprogrammeerd

```
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)))
```

1. 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:

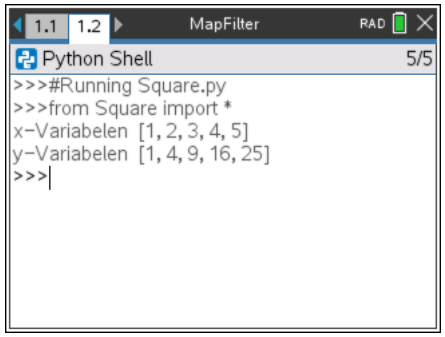
```
kwadraat=[x**2 for x in range(0,11) if x%2==0].
```

2. 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
```



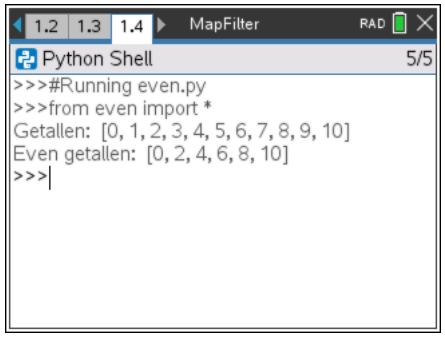
We passen deze functie toe op de lijst [1,2,3,4,5]. Om een nieuwe lijst te creëren moet het map()-statement in combinatie met list() gebruikt worden.

```
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)
```

3. 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
>>>
```

4. Numerieke methodes

4.1. Iteratieve banen

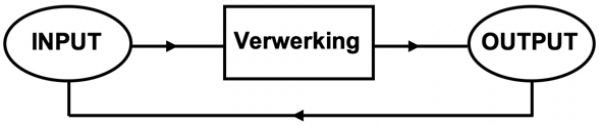
iteratie

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:

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

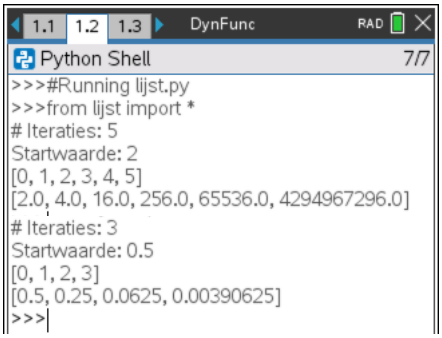
Waarbij F^n betekent dat F n-keer wordt uitgevoerd. Deze functie F noemt men de iteratiefunctie.

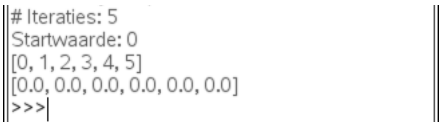
Voor een startwaarde x_0 genereert een iteratieproces een rij $x_0, x_1, x_2, x_3, ..., x_n, ...$.

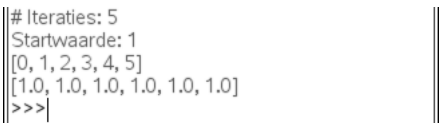
Deze rij noemt men de baan behorende bij de startwaarde x_0.

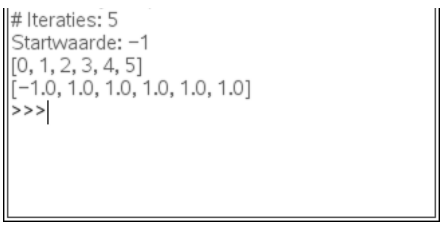
We schrijven een programma dat de banen berekent voor $F(x) = x^2$ en vervolgens visueel/grafisch voorstelt.

```
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.

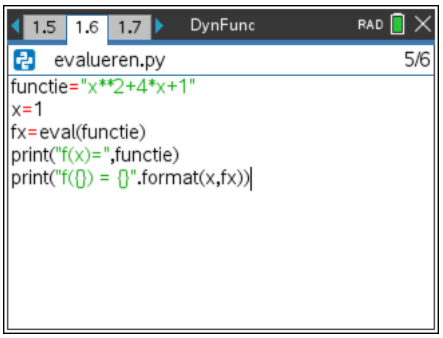
We noteren dit als volgt: $F^n(x) \rightarrow 0$ voor $n \rightarrow +\infty$.

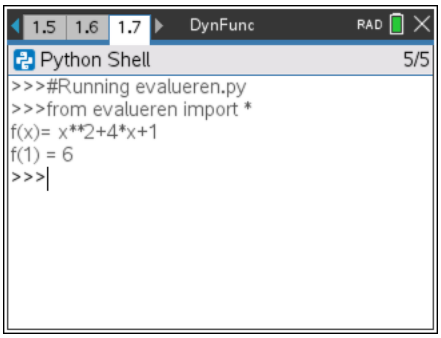
Enkele speciale banen:

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

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

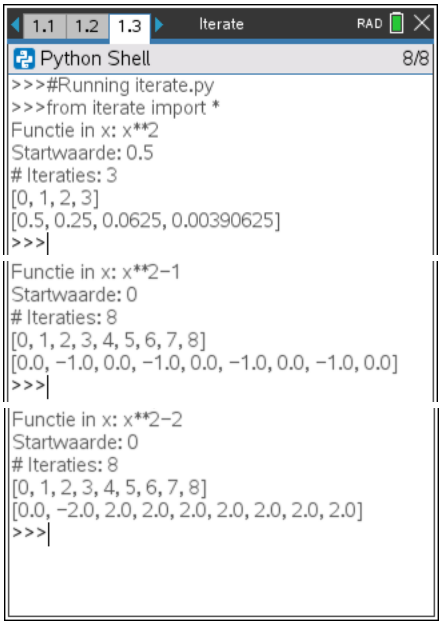
We maken a.h.v. deze code een functie iterate() met als argumenten de functie, de startwaarde en het aantal iteraties. Hiervoor gebruiken o.a. het eval()-statement. Een verschil tussen MicroPython en Python is dat in MicroPython eval() geen gebruik kan maken van lokale variabelen.





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)
```

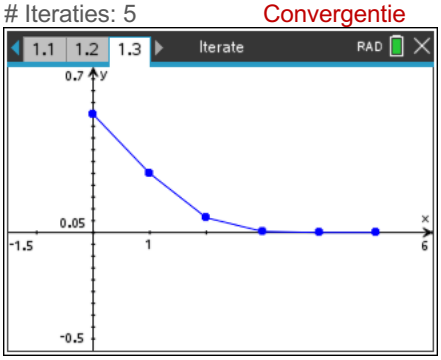


Gebruikmakend van het store_list()-statement van de TI System-module kunnen we de iteratie visueel voorstellen als een scatter plot in Graphs

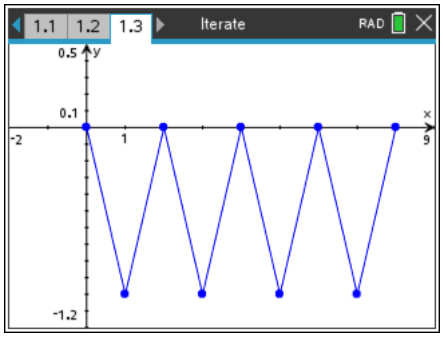
```
from ti_system import *
……
store_list("index",index)
store_list("iteratie",iteratie)
```

Enkele voorbeelden

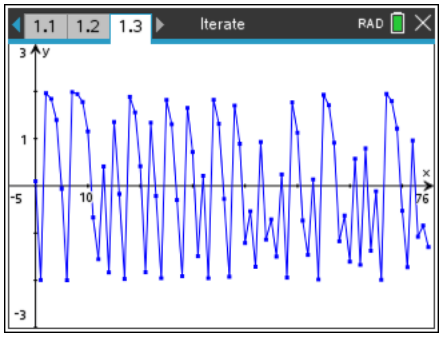
Functie in x: x**2
Startwaarde: 0.5
Iteraties: 5 Convergentie



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 $F(x) = x$.

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

  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 $F^n(x) = x$.

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

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

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

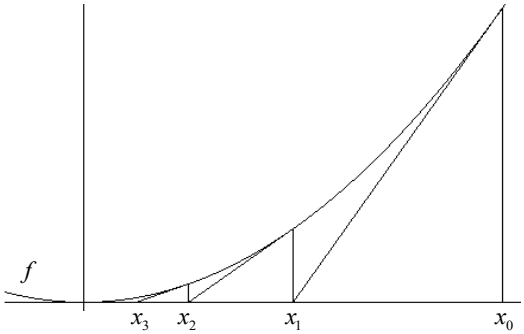
4.2. Newton-Raphson

De methode van Newton-Raphson is een iteratieve numerieke benaderingmethode 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 $x_0, x_1, x_2, x_3, x_4, \ldots$ 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 $x = x_n - \dfrac{f(x_n)}{f'(x_n)}$.

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

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

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)
```

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

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

```
Python Shell 18/18
>>>#Running newton.py
>>>from newton import *
Functie in x: x**2-1
Startwaarde: 2
# Iteraties: 6
NEWTON-RAPHSON
De benadering van een nulpunt van f(x)=x**2-1 in de buurt van x=2.0 is x=1.0
[2.0, 1.25, 1.025, 1.00030487804878, 1.000000046498311, 1.0, 1.0]
>>>
>>>#Running newton.py
>>>from newton import *
Functie in x: x**2-1
Startwaarde: -2
# Iteraties: 6
NEWTON-RAPHSON
De benadering van een nulpunt van f(x)=x**2-1 in de buurt van x=-2.0 is x=-1.0
[-2.0, -1.25, -1.025, -1.00030487804878, -1.000000046498311, -1.0, -1.0]
>>>
```

```
Python Shell 17/17
>>>#Running newton.py
>>>from newton import *
Functie in x: x**2+2*x-8
Startwaarde: 0
# Iteraties: 6
NEWTON-RAPHSON
De benadering van een nulpunt van f(x)=x**2+2*x-8 in de buurt van x=0.0 is x=2.0
[0.0, 4.0, 2.4, 2.023529411764706, 2.000091558691274, 2.000000001349555, 2.0]
>>>#Running newton.py
>>>from newton import *
Functie in x: x**2+2*x-8
Startwaarde: -3
# Iteraties: 5
NEWTON-RAPHSON
De benadering van een nulpunt van f(x)=x**2+2*x-8 in de buurt van x=-3.0 is x=-4.0
[-3.0, -4.25, -4.009615384615385, -4.000015358812481, -4.000000000037478, -4.0]
>>>
```

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.

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.

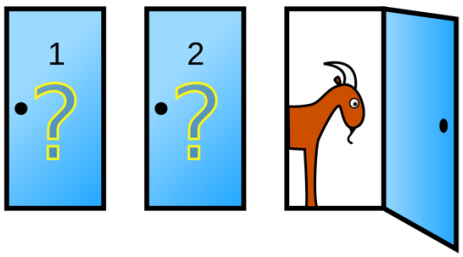
b. Schrijf een programma, gebruikmakend van deze functie, dat alle palindroomgetallen afdrukt tussen 2000 en 3000.

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?

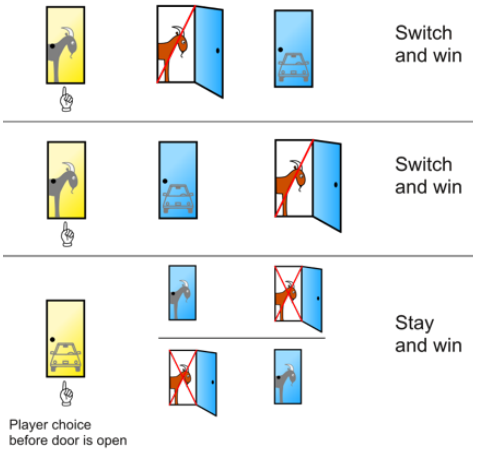
Origineel heet deze brain teaser The Monty Hall problem, gebaseerd op de Amerikaanse TV show Let's make a chance met als gastheer Monty Hall. Het probleem werd voorgelegd aan The America Statistician (een wetenschappelijke academische magazine) in 1975.

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 $n \geq 0$) macht van een getal a berekent.

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

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

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 $a^2 = 2$.

1. 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
```

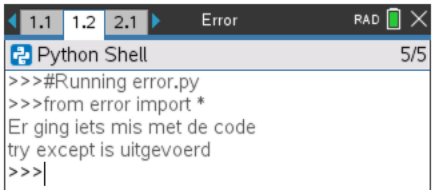
2. try & except

De statements try en except doen het volgende:

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

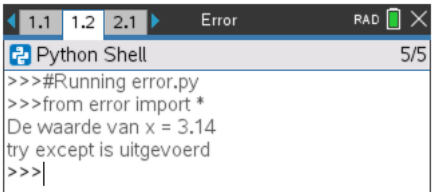
We verduidelijken even deze structuur. Bij het runnen van de onderstaande code krijg je geen error-melding.
try: probeert de code uit te voeren en in het geval van een error wordt de code in het except:-blok uitgevoerd.

```
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.

3. 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.

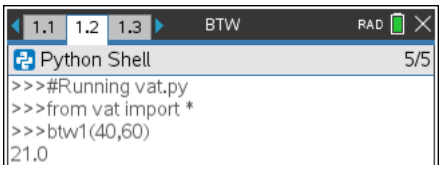
```
while True:
    getal=input("Getal n met 0 ≤ n ≤ 5: ")
    try:
        n=int(getal)
    except ValueError:
        print("VERKEERDE INPUT - Probeer opnieuw!")
        continue
    if n < 0 or n > 5:
        print("BUITEN DE GRENZEN - Probeer opnieuw!")
        continue
    break

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

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

4. *args

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

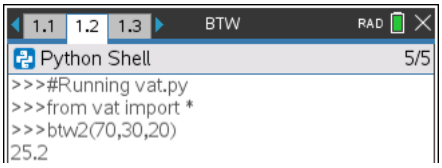
```
def btw1(a,b):
    return sum((a,b))*0.21
```



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

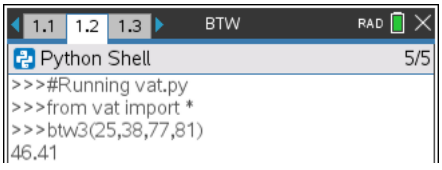
Een manier om dit te doen is het aantal argumenten te verhogen en een standaard waarde toe te kennen aan de argumenten.

```
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 asterisk, *, 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
```



5. 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:

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

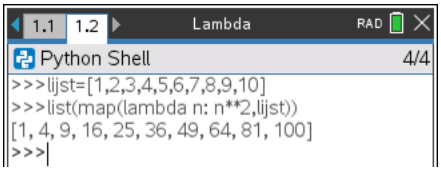
```
def square(n):
    kwad=n**2          <------------>   lambda num : num**2
    return kwad
```

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

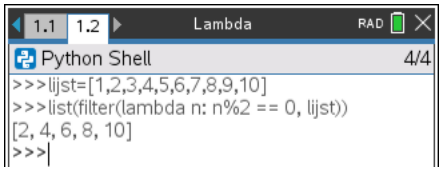
```
square = lambda num: num**2
```

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))



6. 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"))
```

7. Opdrachten

7.1. 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 )
```

7.2. 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:

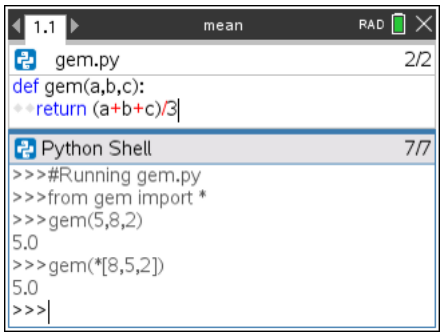
- na iedere uitvoering de optie om verder te stoppen (0) of verder te spelen (1),
- het aantal experimenten wordt bij gehouden, aantal,
- het aantal keren gewonnen wordt bijgehouden, winst.

Benader/simuleer de kans op winst voor beide experimenten.

7.3. 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.

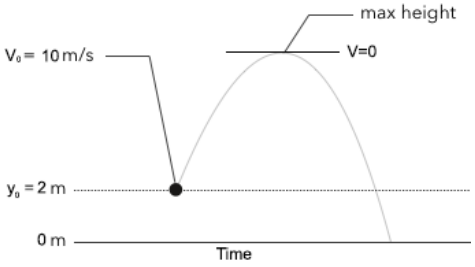
Omgekeerd kan een lijst (of tuple) voorafgegaan door een asteriks, *, gebruikt worden als de argumenten van een functie.



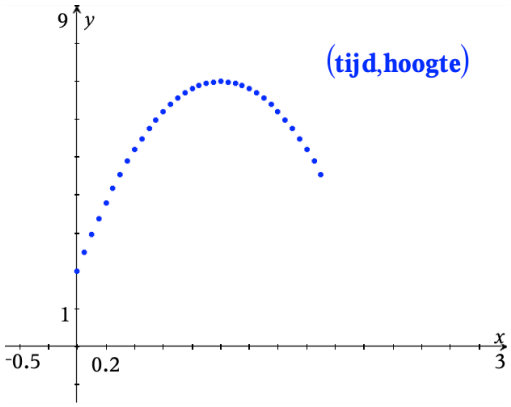
7.4. Gravitatie

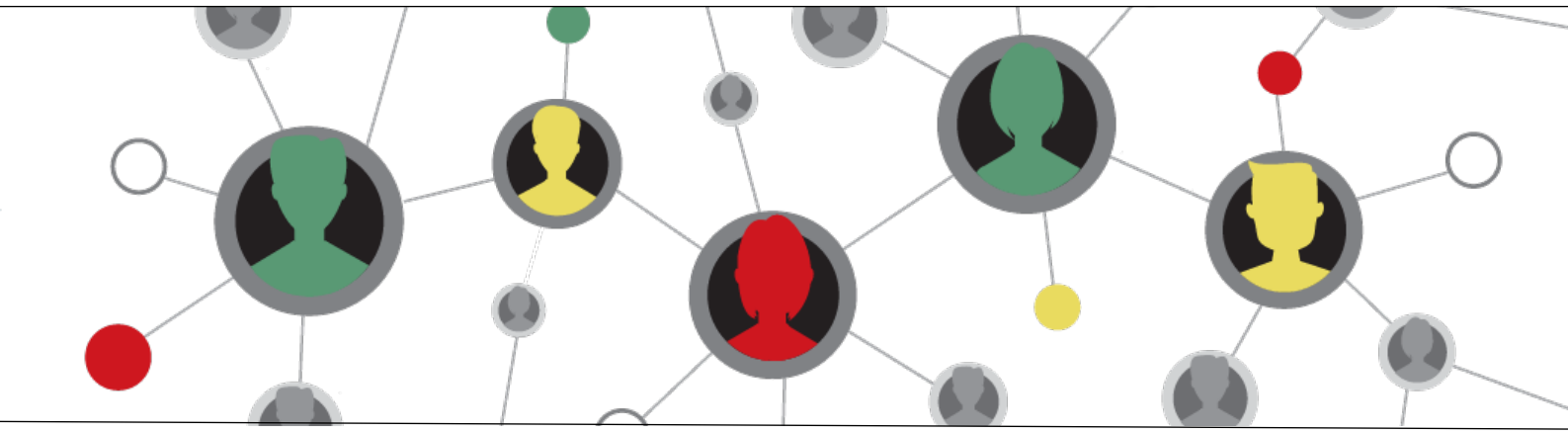
Schrijf een programma om de maximale hoogte te vinden van een bal die recht omhoog wordt gegooid van af een hoogte van 2 meter met een beginsnelheid van 10 m/s.

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



- 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?





T³ NEDERLAND T³ VLAANDEREN

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

Texas Instruments