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.

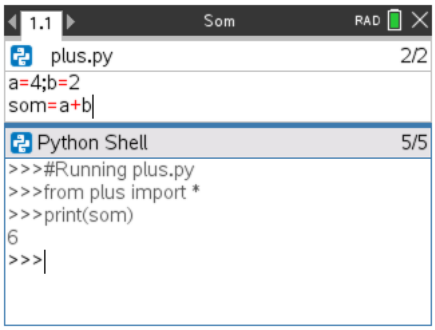
De Python-syntax van een functie ziet er als volgt uit:

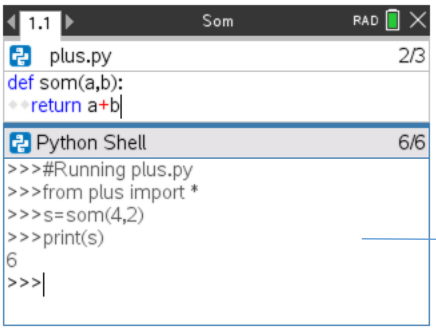
```
def function(argument):
    block
```

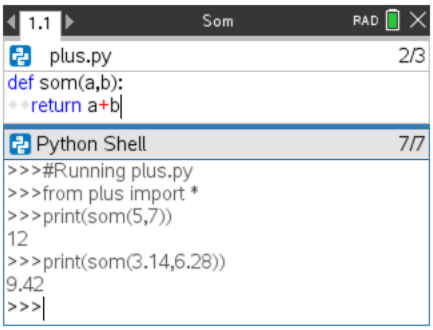
We illustreren de syntax met het voorbeeld de som van twee getallen. Het return-statement laat toe een functie een resultaat te returneren, een resulaat dat b.v. bewaard kan worden als een variabele.

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

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



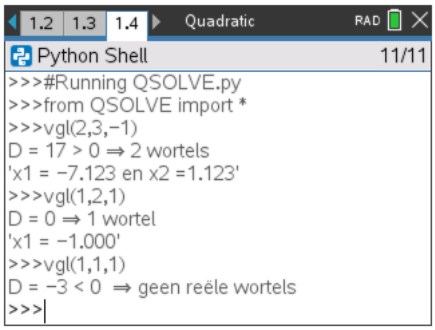




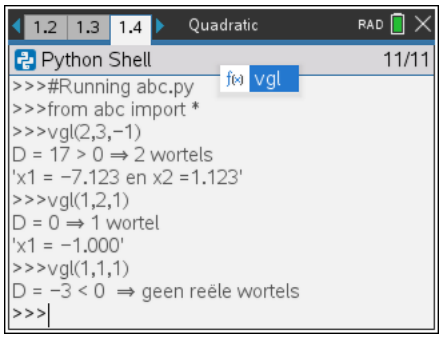
Voorbeeld 1 – De abc-formule

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.

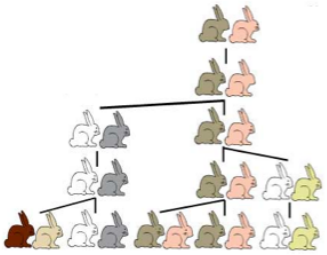


Voorbeeld 2 – Fibonacci & De Gulden Snede

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.

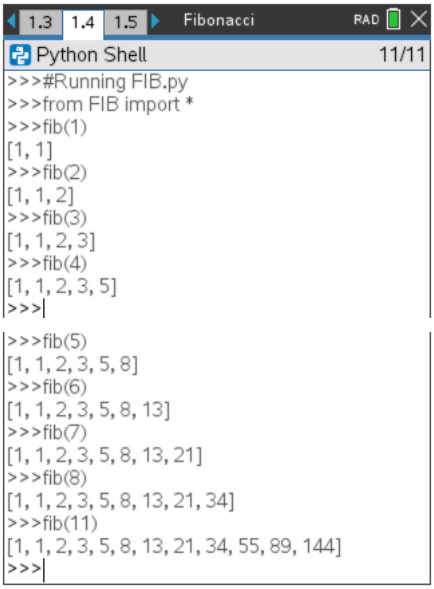
De rij van Fibonacci wordt als volgt op gebouwd:

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

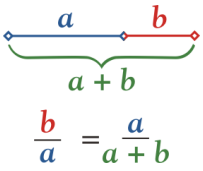
Men kan bewijzen dat: $F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$.

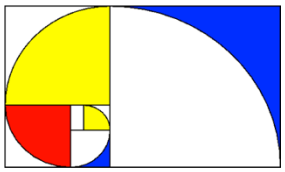
Het volgende programma berekent de rij van Fibonacci:

```
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.



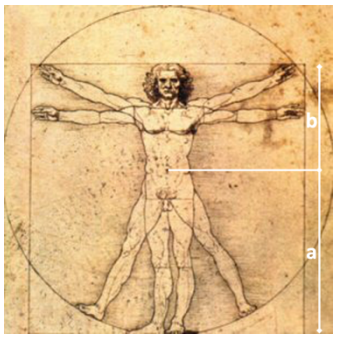


Deze verhouding wordt ook wel het gouden getal genoemd:

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

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:

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

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

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

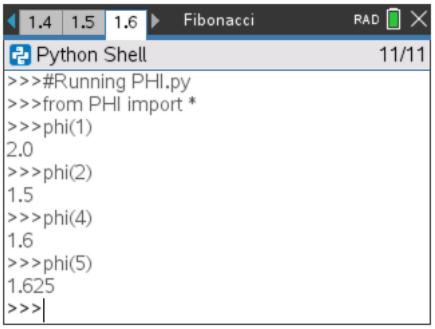
$$\varphi \approx 1 + \frac{1}{1+\frac{1}{1+\frac{1}{1+1}}} = \frac{8}{5} = 1.6$$

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

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

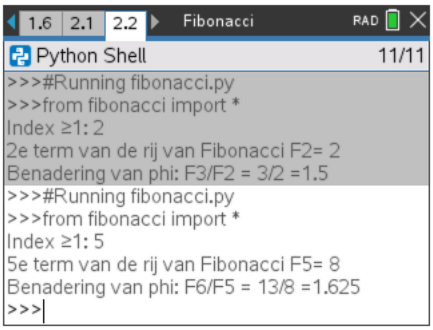
Voor het benaderen van φ door de verhouding van twee opeenvolgende Fibonacci getallen gebruiken we zojuist gedefinieerde functie fib() uit het FIB.py programma. We starten de code met from FIB import *.

```
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)))
```

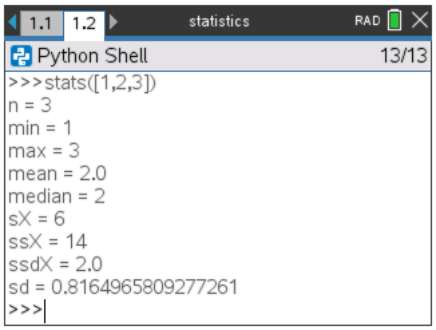


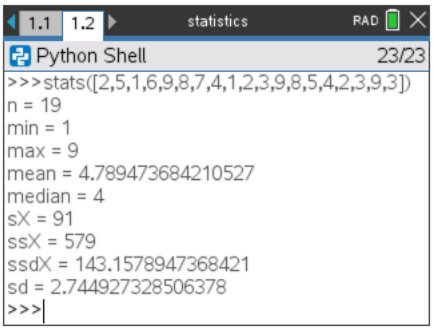
Voorbeeld 3 – Statistische kengetallen

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.

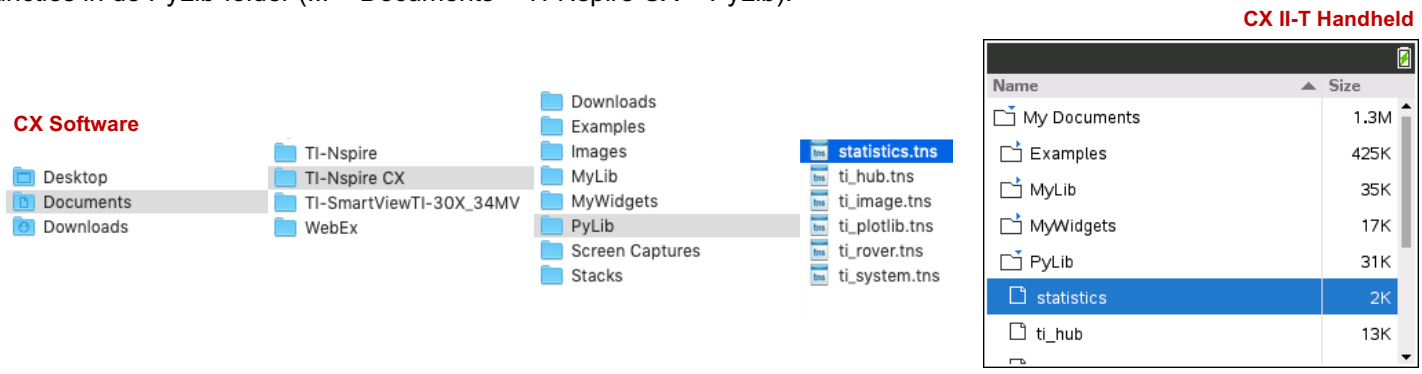
```
def sx(list):
    total=sum(list)
    return total

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
```

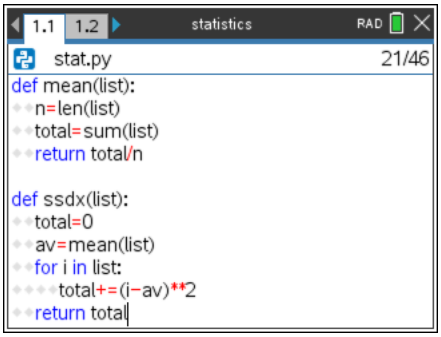




Om deze functies te gebruiken als een zelf gedefinieerde module, plaats je het tns-document met het programma met de functies in de PyLib-folder (... > Documents > TI-Nspire CX > PyLib).

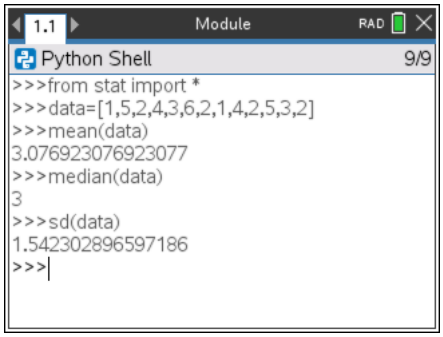


De statistische functies gedefinieerd in het document statistics.tns kunnen nu gebruikt worden na een import van het betreffende programma stat.py in ieder document:



stat-functies

in doc Module



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.

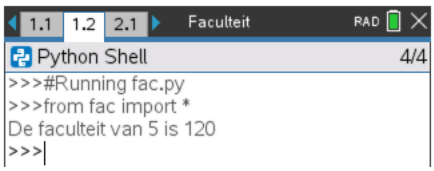
Met behulp van recursieve functies kan het probleem worden onderverdeeld in sub-problemen die eenvoudig kunnen worden opgelost. Maar het is soms moeilijker om de logica van recursieve functie te volgen.

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))
```

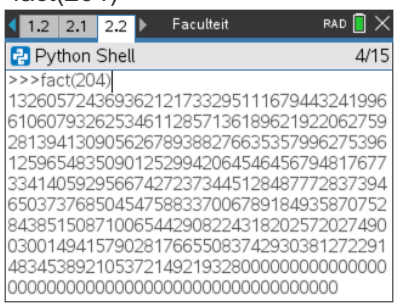


Met recursie

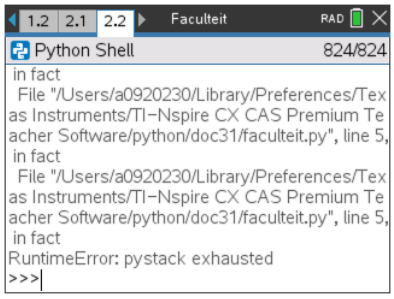
```
def fact(n):
    if n==1:
        return 1
    else:
        return (n*fact(n-1))

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.

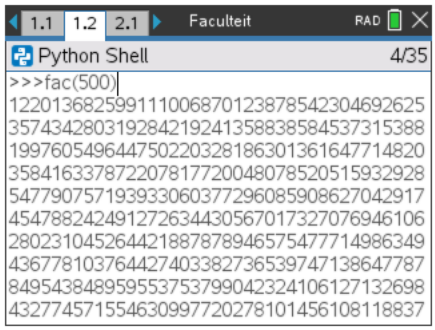
fact(204)

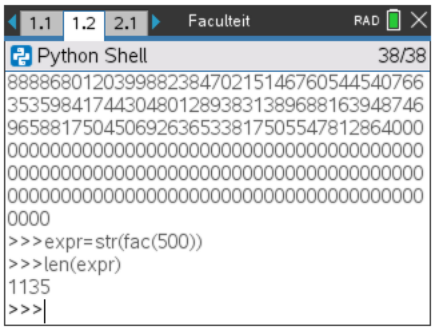


fact(205)



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.




1.1 1.2 2.1 Faculteit RAD
Python Shell 38/38
8888680120399882384702151467605445407663535984174430480128938313896881639487469658817504506926365338175055478128640000000000

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)))
```