

Inleiding Programmeren

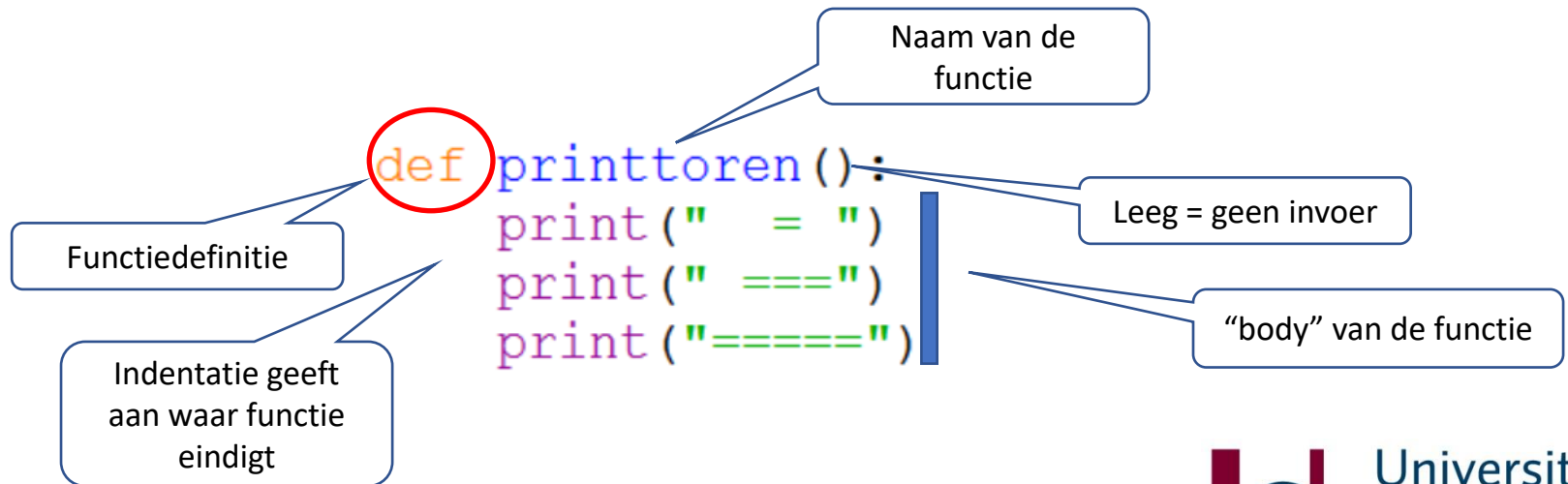
Functies en lijsten

Toon Calders

toon.calders@uantwerpen.be

Functies

- Sommige stukken code willen we op meerdere plaatsen hergebruiken
 - Net zoals “print” en “input”.
- We kunnen dan een *functie* definieren.
 - “mini-programma” met input en output



Functies

```
def printtoren():  
    print("  = ")  
    print(" ===")  
    print("=====")
```

```
print("1 toren:")  
printtoren()
```

```
print("2de toren:")  
printtoren()
```

```
1 toren:  
  =  
  ===  
=====
```

```
2de toren:  
  =  
  ===  
=====
```

Voorbeeld: Turtle-street

```
def huisje():  
    turtle.down()  
    turtle.forward(100)  
    turtle.left(90)  
    turtle.forward(100)  
    turtle.left(45)  
    turtle.forward(math.sqrt(5000))  
    turtle.left(90)  
    turtle.forward(math.sqrt(5000))  
    turtle.left(45)  
    turtle.forward(100)  
    turtle.up()  
    turtle.left(90)
```

```
def street():  
    nr=0  
    while nr<6:  
        huisje()  
        turtle.forward(100)  
        nr=nr+1  
    turtle.right(180)  
    turtle.forward(600)  
    turtle.right(180)
```

```
turtle.up()  
turtle.goto(-400,250)  
st=0  
while st<3:  
    street()  
    turtle.right(90)  
    turtle.forward(200)  
    turtle.left(90)  
    st=st+1
```

Functies

- Functies kunnen overal gedefinieerd worden
 - Maak kan pas gebruikt worden **nadat** ze gedefinieerd is.
- Functies kunnen hergedefinieerd worden (definieer gewoon opnieuw)

Echter!

Afspraak: functies worden steeds bovenaan in het programma gedefinieerd, en eens gedefinieerd wordt hun definitie niet meer gewijzigd

Functies met invoer en uitvoer

- Tot nu toe: functie heeft geen invoer en produceert geen uitvoer (behalve op scherm)
 - Input: via parameters

```
def toren(hoogte):  
    s=""  
    for i in range(hoogte):  
        s=s+" "*(hoogte-i)+"="*(2*i+1)+"\n"  
    return s
```

```
print(toren(5))
```

Output: via return

"=" * 5 = "=====

"\n" betekent: start
nieuwe lijn

Functies met invoer en uitvoer

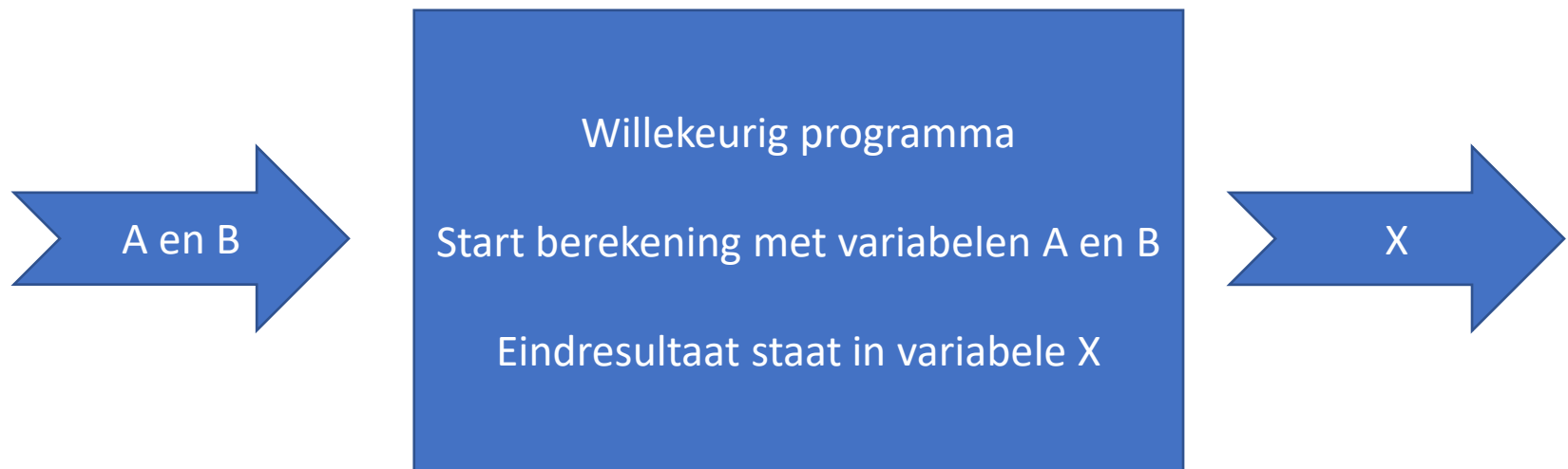
- Tot nu toe: functie heeft geen invoer en produceert geen uitvoer (behalve op scherm)
 - Input: via parameters
 - Output: via return

```
def toren(hoogte):  
    s=""  
    for i in range(hoogte):  
        s=s+" "* (hoogte-i) + "="* (2*i+1) + "\n"  
    return s
```

```
print(toren(5))
```

```
      =  
     ==  
    ===  
   ====  
  =====
```

Functies met invoer en uitvoer



Functies met invoer en uitvoer

```
def functienaam(A, B) :
```

Willekeurig programma

Start berekening met variabelen A en B

Eindresultaat staat in variabele X

```
return X
```

Functies: nomenclatuur

parameter

```
def GGD (A, B) :  
    while A!=0:  
        while A>=B:  
            A = A - B  
        if A!=0:  
            temp=A  
            A=B  
            B=temp  
  
    return (B)
```

Functie-definitie

```
X=int(input("Geef het eerste getal: "))  
Y=int(input("Geef het tweede getal: "))  
print(GGD(X, Y))
```

functieaanroep

argument

Mogelijke fouten: parameters vergeten

```
>>> def GGD(A,B):  
    while A!=0:  
        while A>=B:  
            A = A - B  
        if A!=0:  
            temp=A  
            A=B  
            B=temp  
    return(B)
```

```
>>> print(GGD())
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#9>", line 1, in <module>
```

```
    print(GGD())
```

```
TypeError: GGD() missing 2 required positional arguments: 'A' and 'B'
```

Mogelijke fouten: return vergeten

```
>>> def GGD(A,B):  
    while A!=0:  
        while A>=B:  
            A = A - B  
        if A!=0:  
            temp=A  
            A=B  
            B=temp
```

```
>>>  
>>> print(GGD(245,113))  
None
```

Mogelijke fouten: parameters hebben verkeerd type

```
>>> def GGD(A,B):  
    while A!=0:  
        while A>=B:  
            A = A - B  
        if A!=0:  
            temp=A  
            A=B  
            B=temp  
    return(B)
```

```
>>> print(GGD("Python","fun!!!"))  
Traceback (most recent call last):  
  File "<pyshell#16>", line 1, in <module>  
    print(GGD("Python","fun!!!"))  
  File "<pyshell#15>", line 4, in GGD  
    A = A - B  
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

Voorbeeld: Turtle street met parameters

- Voeg parameters toe:
 - Grootte huis
 - Aantal huizen
 - Aantal straten

Functies: voorbeelden

```
def priem(X):  
    if X<2:  
        return False  
    d=2  
    while d<X:  
        if X%d==0:  
            return False  
        d=d+1  
    return True
```

```
def herhaal(s,n):  
    result=""  
    while n>0:  
        result=result+s  
        n=n-1  
    return result
```

Return breekt
onmiddellijk de functie
af en kan eender waar
geplaats worden binnen
de functie

Functies die functies aanroepen

```
def priem(X):  
    if X<2:  
        return False  
    d=2  
    while d<X:  
        if X%d==0:  
            return False  
        d=d+1  
    return True
```

```
def priemOntbinding(X):  
    d=2  
    while X>1:  
        if priem(d):  
            if X%d==0:  
                print(d)  
                X=X/d  
        d=d+1
```

Aanroep van een andere
functie

Vlugge vraag

- Wat loopt er mis bij deze functie? Hoe maken?

```
def priemOntbinding(X):  
    d=2  
    while X>1:  
        if priem(d):  
            if X%d==0:  
                print(d)  
                X=X/d  
        d=d+1
```

```
>>> priemOntbinding(21)  
3  
7
```

Gaat prima!

```
>>> priemOntbinding(45)  
3  
5  
|
```

Loopt vast!

Variabelen en functies

- Functies hebben **niet automatisch toegang** tot variabelen in het hoofdprogramma
- Omgekeerd heeft het hoofdprogramma **geen toegang** tot de variabelen van de functie
- Het gebied van het programma waarin een variabele “kan gezien worden”, noemen we de **scope** van de variabele

Afspraak: functies communiceren enkel met het hoofdprogramma via de parameters en de return waarde.

Modules

- Je kan zelfs functies uit andere bestanden aanroepen:

```
from euclides import GGD  
  
print(GGD(134, 3428))
```

```
import euclides  
  
print(euclides.GGD(134, 3428))
```

- Python moet deze bestanden wel kunnen vinden (bvb. in zelfde map)
- Merk op: dit deden we al met turtle!
 - Andere handige module: math

Oefening: bevriende getallen

Van twee verschillende [natuurlijke getallen](#) a en b wordt gezegd dat ze **bevriend** zijn als de som van de *echte* [delers](#) van het getal a (a zelf niet, maar 1 wel) gelijk is aan het getal b , terwijl de som van echte delers van b gelijk is aan het getal a .

Een sinds de oudheid bekend paar bevriende getallen is (220, 284):

(som van echte delers van 220) = $1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$

(som van echte delers van 284) = $1 + 2 + 4 + 71 + 142 = 220$

https://nl.wikipedia.org/wiki/Bevriende_getallen

- Ontwerp een algoritme dat alle bevriende getallen vindt met $a, b < 2000$

Nieuw data type: Lijsten

- Een lijst is een zogenaamde “container”
 - Bevat meerdere waarden tegelijk
- Gebruik van een lijst:
 - Initialisatie: $L = [\text{elementen gescheiden door } ',']$
 - Lengte: $\text{len}(L)$
 - Element op positie i : $L[i]$ ← we beginnen te tellen vanaf 0 !
 - Element toevoegen: $L.\text{append}(\text{element})$
 - 2 lijsten samenvoegen: $L1 + L2$
 - Sublijst van s tot $t-1$: $L[s:t]$

Voorbeeld: lijst

```
L = [1, 2, 3, "a", "b"]
```

```
print(L)
```



[1, 2, 3, 'a', 'b']

```
i=0
```

```
while i<len(L):
```

```
    print(L[i])
```

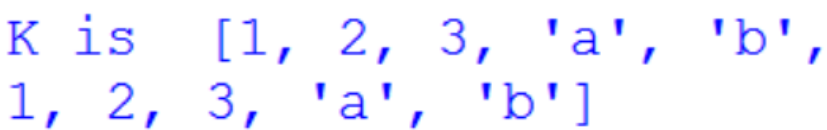
```
    i=i+1
```



1
2
3
a
b

```
K = L+L
```

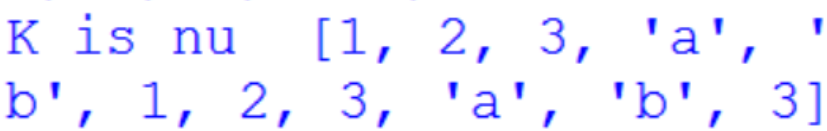
```
print("K is ",K)
```



K is [1, 2, 3, 'a', 'b', 1, 2, 3, 'a', 'b']

```
K.append(3)
```

```
print("K is nu ",K)
```



K is nu [1, 2, 3, 'a', 'b', 1, 2, 3, 'a', 'b', 3]

Lees een lijst getallen in

- Lees een lijst getallen in
 - Afsluiten met leeg antwoord

Lees een lijst getallen in

- Lees een lijst getallen in
 - Afsluiten met leeg antwoord

```
ans=" "  
getallen=[]  
while ans!="":  
    ans=input("Geef volgend getal of enter om te stoppen: ")  
    if ans!="":  
        getallen.append(float(ans))
```


For Loop

for *variabele* in *lijst*:
 uit te voeren code

- Herhaalt *uit te voeren code* zoveel maal als er elementen in *lijst* zijn. In elke iteratie wordt *variabele* gelijk gemaakt aan het volgende element in de lijst.

Voorbeeld: For Loop

```
i=0
L=["Python", "is", "fun", "!"]
for w in [1,1,3,4]:
    j=0
    while j<w:
        print(L[i])
        j=j+1
    i=i+1
```

Voorbeeld: For Loop

```
i=0
L=["Python", "is", "fun", "!"]
for w in [1,1,3,4]:
    j=0
    while j<w:
        print(L[i])
        j=j+1
    i=i+1
```

Python
is
fun
fun
fun
!
!
!
!

Bereken gemiddelde en standaardafwijking

- Lijst getallen ingelezen van gebruiker
 - Code op eerdere slide
- Bereken gemiddelde en (sample) standaardafwijking

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$$

```
# mean and standard deviation

ans=" "
getallen=[]
while ans!="":
    ans=input("Geef volgend getal of enter om te stoppen: ")
    if ans!="":
        getallen.append(float(ans))

som=0
for x in getallen:
    som=som+x

gemiddelde=som/len(getallen)

stdev=0
for x in getallen:
    stdev=stdev+(x-gemiddelde)**2

stdev=(stdev/(len(getallen)-1))**0.5

print("Steekproef gemiddelde en standaardafwijking zijn: ",gemiddelde,stdev)
```

Speciale lijsten

- `range(5) = [0,1,2,3,4]`
- `range(3,5) = [3,4]`

(in werkelijkheid is “range” niet echt een lijst, maar voorlopig kan je range als een lijst beschouwen)

Voorbeeld: priemgetal

```
# Bepaal of een gegeven getal priem is
```

```
X=int(input("Geef het te testen getal: "))
```

```
priem=True
```

```
for i in range(2,X):  
    if X%i==0:  
        priem = False
```

[2,3,...,X-1]

```
if priem:  
    print(X, "is priem")  
else:  
    print(X, "is niet priem")
```

Oefening: Zeef van Eratosthenes

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

Oefening: Zeef van Eratosthenes

- Wat is input? Wat is output?
- Welke *data-structuur* gaan we gebruiken?
 - Heel belangrijk! Een goede data-structuur bespaart je veel programmeerwerk!
- Welke manipulaties op de data structuur zijn nodig?
 - Algoritme
- Nu pas: implementatie
 - Test tussentijds!

Oefening: Zeef van Eratosthenes

```
# Bereken priemgetallen mbv de Zeef van Erat
```

```
laatste=20      # grootte van de zeef
```

```
Zeef = ['Priem']*laatste
```

```
Zeef[0]='Geen priem' # 0 en 1 zijn geen priemgetallen
```

```
Zeef[1]='Geen priem'
```

```
for i in range(2,laatste): # voor elk getal
```

```
    if Zeef[i]=='Priem': # als het priem is, schrap veelvoud
```

```
        veelvoud=i*i # eerste veelvoud te beschouwen is i**2
```

```
        while veelvoud < laatste:
```

```
            Zeef[veelvoud]='Geen priem'
```

```
            veelvoud+=i
```

```
# print de output van het programma
```

```
for i in range(laatste):
```

```
    print("Het getal ",i," is ",Zeef[i])
```

Maak een lijst door de lijst
['Priem'] zoveel te
herhalen als de waarde in
laatste