Hoofdstuk 8: Functies en parameters

Leerdoelen:

- Het nut van functies
- Eigen functies creëren
- Parameters en argumenten gebruiken
- Waardes uit functies retourneren middels return
- Functie benamingen
- Commentaar in functies
- Variabele scope en levensduur
- Lokale en globale variabelen
- Het gebruik van functies om grip te krijgen op complexiteit

Mededelingen:

- Mailbox vol met repl.it mail?
  - Zet mail ontvangen uit in je profile

- De planner staat op **course.cs.ru.nl/pythonVO**
  (deze link staat ook op Montiplaza)

Eigen functies maken: waarom?

Opdelen in samenhangende delen (=functies),
Maakt programmeren overzichtelijker:
focus even op 1 ding maken & testen

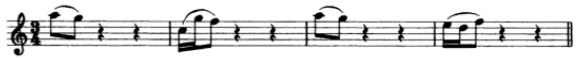
Waarom functies?

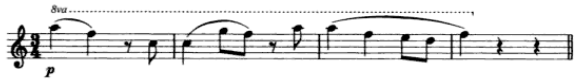
- Code hergebruiken
- Betere structuur
- Minder fouten maken
- Makkelijker fouten opsporen

Functies met parameters voor deeltaken

- Grote of complexe taken deel je op in deeltaken.
- Elke deeltaak los je apart van de rest op.
- Een functie kun je vaker aanroepen.
- Dat scheelt heel veel tikwerk.

- Soms lijken twee functies veel op elkaar, maar verschillen ze in kleine details. Je kan dan **parameters** gebruiken om de details aan te geven





- Hoe dat werkt zien we nu.
- We gaan **functies** schrijven met **parameters** dat een kinderliedje (met veel herhalingen) voor ons kan ‘zingen’.

De wielen van de bus….

De wielen van de bus gaan rond en rond
Draaien rond, draaien rond
De wielen van de bus gaan rond en rond
Als de bus gaat rijden

De deuren van de bus gaan open en dicht
Open en dicht, open en dicht
De deuren van de bus gaan open en dicht
Als de bus gaat rijden

De wissers van de bus gaan heen en weer
Heen en weer, heen en weer
De wissers van de bus gaan heen en weer
Als de bus gaat rijden

werkblad

Alle tekst afdrukken?
Allemaal overtikken?
… ook voor alle 120 andere couplets, of..
patroon ontdekken
Probleem makkelijker maken

Opgave 8.2.3

De wielen van de bus….

De wielen van de bus gaan rond en rond
Draaien rond, draaien rond
De wielen van de bus gaan rond en rond
Als de bus gaat rijden

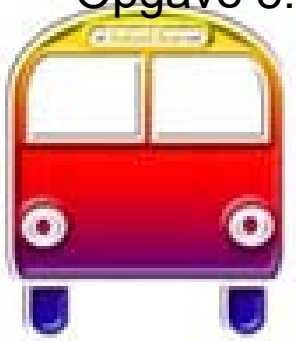
De deuren van de bus gaan open en dicht
Open en dicht, open en dicht
De deuren van de bus gaan open en dicht
Als de bus gaat rijden

De wissers van de bus gaan heen en weer
Heen en weer, heen en weer
De wissers van de bus gaan heen en weer
Als de bus gaat rijden

Alle tekst afdrukken?
Allemaal overtikken?
… ook voor alle 120 andere couplets, of..
patroon ontdekken
Probleem makkelijker maken

Opgave 8.2.3

De wielen van de bus….



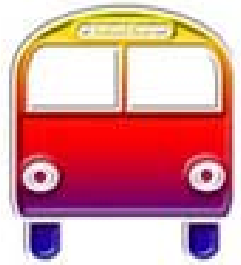
De wielen van de bus gaan rond en rond
Rond en rond, rond en rond
De wielen van de bus gaan rond en rond
Als de bus gaat rijden

De deuren van de bus gaan open en dicht
Open en dicht, open en dicht
De deuren van de bus gaan open en dicht
Als de bus gaat rijden

Onderstreep in het bovenste couplet wat precies hetzelfde is als in het onderste couplet.

Opgave 8.2.3

De wielen van de bus….



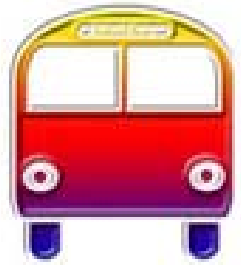
De wielen van de bus gaan rond en rond
Rond en rond, rond en rond
De wielen van de bus gaan rond en rond
Als de bus gaat rijden

De deuren van de bus gaan open en dicht
Open en dicht, open en dicht
De deuren van de bus gaan open en dicht
Als de bus gaat rijden

… herhalingen in beide coupletten onderstreept
... Dit kun je met een gewone `print("...")` afwerken

Opgave 8.2.3

De wielen van de bus….



De deuren van de bus gaan open en dicht
Open en dicht, open en dicht
De deuren van de bus gaan open en dicht
Als de bus gaat rijden

deel

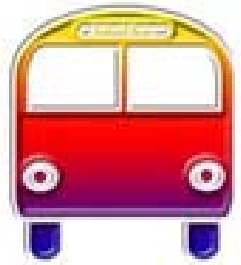
actie

Blijft er iets in het bovenste couplet over wat zich herhaalt?
Omcirkel alle voorkomens.
Wat voor iets is dit? Geef het een naam.

Blijft er nog iets over in het bovenste couplet wat zich herhaalt?
Teken een **rechthoek** om alle voorkomens.
Geef het een naam.

Opgave 8.2.3

De wielen van de bus….



De wielen van de bus gaan rond en rond
Rond en rond, rond en rond
De wielen van de bus gaan rond en rond
Als de bus gaat rijden

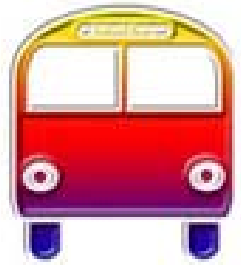
De deuren van de bus gaan open en dicht
Open en dicht, open en dicht
De deuren van de bus gaan open en dicht
Als de bus gaat rijden

deel

actie

Opgave 8.2.3

De wielen van de bus….



De **deel** van de bus gaan **actie**

actie **actie**

De **deel** van de bus gaan **actie**

Als de bus gaat rijden

De **deel** van de bus gaan **actie**

actie **actie**

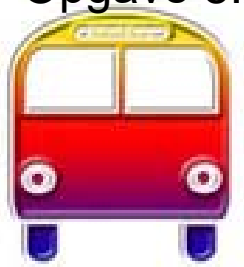
De **deel** van de bus gaan **actie**

Als de bus gaat rijden

Vervang herhalende delen in liedje door de variabele
(dus ipv **wielen** schrijf je **deel**)

Opgave 8.2.3

De wielen van de bus….



Hier maken we een functie definitie van:

```
def zingCouplet( deel , actie ):
        print( "De " + deel + " van de bus gaan + " actie )
        print( actie + "," + actie )
        print( "De " + deel + " van de bus gaan + " actie )
        print( "Als de bus gaat rijden" )
```

Als je nu de functie aanroept:

```
zingCouplet(                ,                )
```

Vult het "deuren" in bij deel (want die staat als eerste)
en "open en dicht" bij actie (want die staat als tweede)

Functies maken en gebruiken

```
# FUNCTIE DEFINITIES
def zingCouplet( deel , actie ):
        print( "De " + deel + " van de bus gaan + " actie )
        print( actie + "," + actie )
        print( "De " + deel + " van de bus gaan + " actie )
        print( "Als de bus gaat rijden" )
        print( "\n" )

# HOOFDPROGRAMMA: AANROEP VAN FUNCTIES
zingCouplet( "wielen", "rond en rond")
zingCouplet( "wissers", "heen en weer")
zingCouplet( "lichten", "aan en uit")
```

Functies maken en gebruiken

LIVE CODING

```
# FUNCTIE DEFINITIES
def zingCouplet( deel , actie ):
        print( "De " + deel + " van de bus gaan + " actie )
        print( actie + "," + actie )
        print( "De " + deel + " van de bus gaan + " actie )
        print( "Als de bus gaat rijden" )
        print( "\n" )

# HOOFDPROGRAMMA: AANROEP VAN FUNCTIES
zingCouplet( "wielen", "rond en rond")
zingCouplet( "wissers", "heen en weer")
zingCouplet( "lichten", "aan en uit")
```

Wanneer gebruik je functies?

- Gebruik je dezelfde code een paar keer???
- Heb je een deja-vu gevoel???

- Niet copy-pasten!
- Maak gebruik van **functies** en **parameters**

Functies definiëren en gebruiken

```
import turtle

turtle.pendown()

# tekenen van een vierkant
turtle.forward(50)
turtle.right(90)
turtle.forward(50)
turtle.right(90)
turtle.forward(50)
turtle.right(90)
turtle.forward(50)


turtle.done()
```

```
import turtle

#Definitie: instructies voor een vierkant
def vierkant():
    turtle.forward(50)
    turtle.right(90)
    turtle.forward(50)
    turtle.right(90)
    turtle.forward(50)
    turtle.right(90)
    turtle.forward(50)

turtle.pendown()
vierkant() #Aanroep: teken de vierkant!

turtle.done()
```

Functies gebruiken

```
import turtle

#Definitie: instructies voor vierkant
def vierkant():
    turtle.forward(50)
    turtle.right(90)
    turtle.forward(50)
    turtle.right(90)
    turtle.forward(50)
    turtle.right(90)
    turtle.forward(50)


turtle.pendown()
vierkant() #Aanroep: teken vierkant!
turtle.right(30)
vierkant() #Aanroep: teken vierkant!

turtle.done()
```

Voordeel van functies:

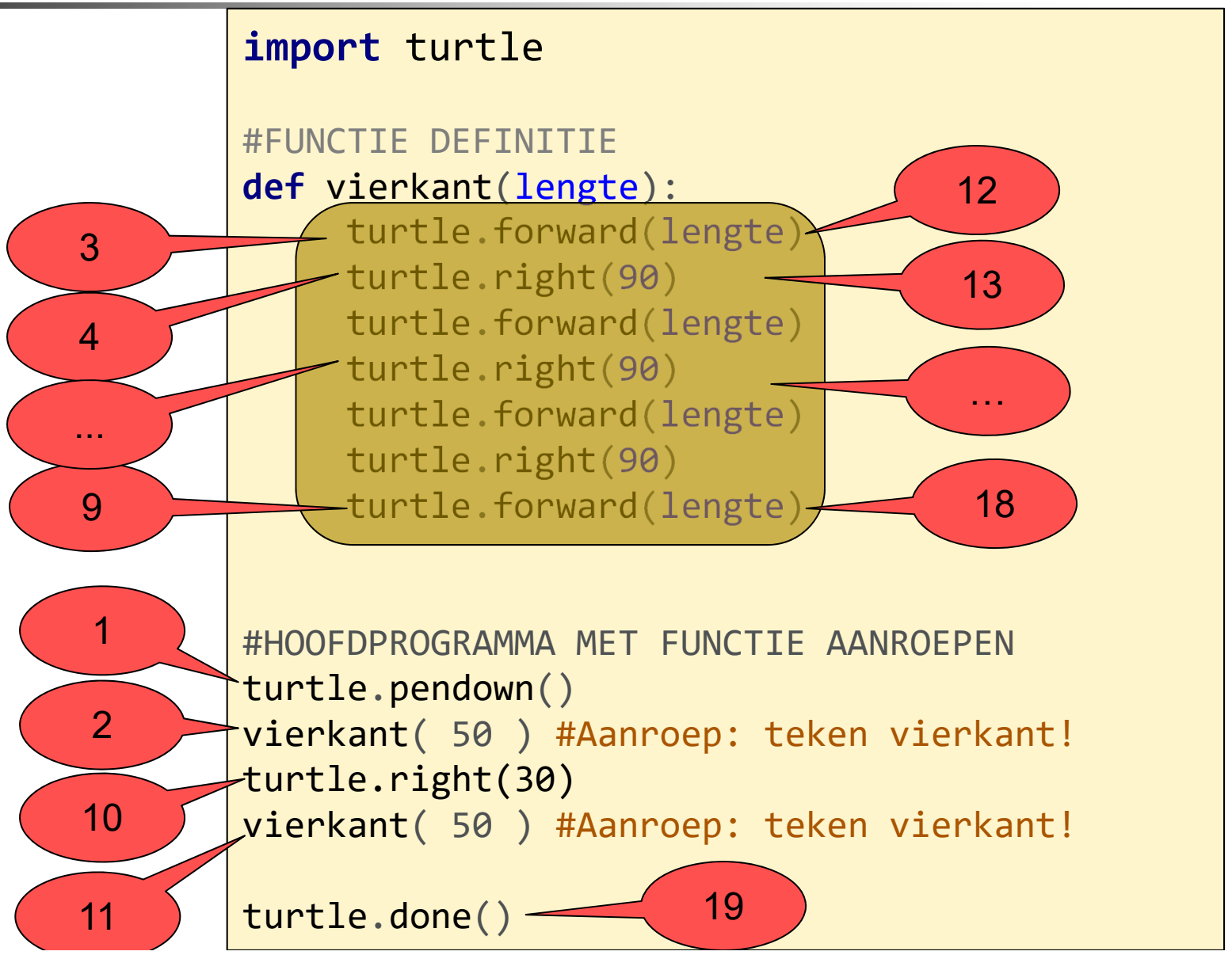
- Je kunt iets tekenen ZONDER bezig te zijn met details
- hergebruik: veel minder code!
- Aanpasbaar op 1 plek

Parameters gebruiken

```
import turtle

turtle.pendown()

# tekenen van een vierkant
turtle.forward(50)
turtle.right(90)
turtle.forward(50)
turtle.right(90)
turtle.forward(50)
turtle.right(90)
turtle.forward(50)


turtle.done()
```

```
import turtle

#Definitie: instructies voor een vierkant
# met een gegeven lengte
def vierkant( lengte ):
    turtle.forward( lengte )
    turtle.right(90)
    turtle.forward( lengte )
    turtle.right(90)
    turtle.forward (lengte )
    turtle.right(90)
    turtle.forward (lengte )

turtle.pendown()

#Aanroep: teken vierkant met lengte 50
vierkant( 50 )

turtle.done()
```

Hoe het programma verloopt



Terugblik: functies met parameters

```
import turtle

#FUNCTIE DEFINITIE
def tekenVierkant( lengte ):
    turtle.forward( lengte )
    turtle.right( 90 )
    turtle.forward( lengte )
    turtle.right( 90 )
    turtle.forward( lengte )
    turtle.right( 90 )
    turtle.forward( lengte )


#HOOFDPROGRAMMA
turtle.pendown()
tekenVierkant(50) #teken een vierkant
turtle.right(30)
tekenVierkant(50) #teken een vierkant

turtle.done()
```

Functie **definiëren**

Functie **aanroepen**

Functie met parameters

Voorbeeld:

```
import turtle

#FUNCTIE DEFINITIE
def tekenLijn(lengte):
     turtle.forward(lengte)

#HOOFDPROGRAMMA MET FUNCTIE AANROEP
turtle.pendown()
tekenLijn( 50 ) #Aanroep: teken lijn van 50

turtle.done()
```

In het algemeen:

```
#FUNCTIE DEFINITIE
def zinnigeNaamInCamelCase(parameter1, parameter2):
    doeIets()

#HOOFDPROGRAMMA MET FUNCTIE AANROEP
zinnigeNaamInCamelCase(waarde1, waarde2)
```

Functie definitie en aanroep

In het algemeen:

```
#FUNCTIE DEFINITIE
def zinnigeNaamInCamelCase(parameter1, parameter2):
    doeIets()

#HOOFDPROGRAMMA MET FUNCTIE AANROEP
zinnigeNaamInCamelCase(waarde1, waarde2)
```

Functie **definiëren**

Functie **aanroepen**

LIVE CODING

Functie met parameters

- Doel: schrijf een functie die de omtrek van een rechthoek uitrekent en afdrukt
- Parameters: krijgt twee getallen mee voor berekening
  - breedte en hoogte
- Berekening: omtrek = 2*breedte + 2*hoogte
- Roep je functie aan.
  - De breedte is 5, de hoogte is 3

```
#FUNCTIE DEFINITIE
def zinnigeNaamInCamelCase(parameter1, parameter2):
    doeIets()

#HOOFDPROGRAMMA MET FUNCTIE AANROEP
zinnigeNaamInCamelCase(waarde1, waarde2)
```

Functie met parameters: oplossing

- Doel: schrijf een functie die de omtrek van een rechthoek uitrekent en afdrukt
- Parameters: krijgt twee getallen mee voor berekening
  - breedte en hoogte
- Berekening: omtrek = 2*breedte + 2*hoogte
- Roep je functie aan.
  - De breedte is 5, de hoogte is 3

```
#FUNCTIE DEFINITIE
def berekenOmtrekRechthoek ( breedte, hoogte ):
    omtrek = 2*breedte + 2*hoogte
    print(omtrek)

#HOOFDPROGRAMMA MET FUNCTIE AANROEP
berekenOmtrekRechthoek(5,3)
```

Functies: Scope en levensduur

- Stel je wilt later dat omtrek nog gebruiken.
  - Hekwerk kost 10E per meter.

```
#FUNCTIE DEFINITIE
def berekenOmtrekRechthoek ( breedte, hoogte ):
    omtrek = 2*breedte + 2*hoogte
    print(omtrek)

#HOOFDPROGRAMMA MET FUNCTIE AANROEP
berekenOmtrekRechthoek(5,3)
kosten = 10*omtrek
print(kosten)
```

Dit kan niet, want ‘omtrek’
bestaat alleen binnen de functie

Scope en levensduur

```
#FUNCTIE DEFINITIE
def berekenOmtrekRechthoek(breedte, hoogte):
    omtrek = 2*breedte + 2*hoogte
    print(omtrek)

#HOOFDPROGRAMMA MET FUNCTIE AANROEP
berekenOmtrekRechthoek(5,3)
kosten = 10*omtrek
print(kosten)
```

omtrek wordt in functie definitie gemaakt,
bestaat alleen in functie,
Na afloop van functie bestaat het niet meer

Scope en levensduur

```
#FUNCTIE DEFINITIE
def berekenOmtrekRechthoek(breedte, hoogte):
    omtrek = 2*breedte + 2*hoogte
    print(omtrek)

#HOOFDPROGRAMMA MET FUNCTIE AANROEP
berekenOmtrekRechthoek(5,3)
kosten_hekwerk = 10*omtrek
print(kosten_hekwerk)
```

omtrek wordt in functie definitie gemaakt,
bestaat alleen in functie,
na afloop van functie bestaat het niet meer

Wil je omtrek toch nog gebruiken?

1. Dan moet de functie deze na afloop teruggeven (retourneren)
   In plaats van print(omtrek) gebruik je return omtrek
2. In hoofdprogramma de teruggegeven waarde opslaan

```
#FUNCTIE DEFINITIE
def zinnigeNaamInCamelCase(parameter1, parameter2):
    doe_iets

#HOOFDPROGRAMMA MET FUNCTIE AANROEPEN
zinnigeNaamInCamelCase(waarde1, waarde2)
```

- Naam *zinnigeNaamInCamelCase*
- Parameters: heeft twee getallen nodig voor berekening
  - breedte en hoogte *(… , ….)*
- Berekening:
  - omtrek = 2*breedte + 2*hoogte

```
def 
    omtrek = 
    return omtrek
```

Gebruik van return bij functie

```
#FUNCTIE DEFINITIE
#deze functie levert de omtrek van een rechthoek op
def omtrekRechthoek ( breedte, hoogte ):
    omtrek = 2*breedte + 2*hoogte
    return omtrek

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
meters_hekwerk = omtrekRechthoek( 5, 3 )
print( "Aan meter hekwerk nodig:", meters_hekwerk )
```

Geef berekende omtrek terug

Sla teruggegeven waarde op in een variabele.
Naam mag je zelf kiezen.

Functie definitie en aanroep

```
#FUNCTIE DEFINITIE
#deze functie levert de omtrek van een rechthoek op
def omtrekRechthoek ( breedte, hoogte ):
    omtrek = 2*breedte + 2*hoogte
    return omtrek

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
meters_hekwerk = omtrekRechthoek( 5, 3 )
print( "Aan meter hekwerk nodig is", meters_hekwerk )
kosten_hekwerk = 10 ters_hekwerk
print( "Kosten voor h is", kosten_hekwerk )
```

Geef berekende omtrek terug

Sla teruggegeven waarde op in een variabele.
Naam mag je zelf kiezen

Functies met return oefenen

Maak nu van Hoofdstuk 8 Eigen Functies: Theorie en opgaven

- Maak eerst Opgave 8.2.4, en dan
- Opgave 8.2.5 Grootste waarde retourneren

Over 5 minuten bespreken we deze samen.

Ben je eerder klaar? Ga verder vanaf Opgave 8.1

Functie grootste

Schrijf nu zelf een functie die twee getallen als parameters binnen krijgt en de grootste oplevert

Roep de functie aan met getallen (4,5) en druk de grootste af.

Voorbeeld:

```
#FUNCTIE DEFINITIE
#deze functie levert de omtrek van een rechthoek op
def omtrekRechthoek ( breedte, hoogte ):
    omtrek = 2*breedte + 2*hoogte
    return omtrek

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
meters_hekwerk = omtrekRechthoek( 5, 3 )
print( “Aan meter hekwerk nodig is", meters_hekwerk )
```

Functie grootste: stap 1

Schrijf nu zelf een functie die twee getallen als parameters binnen krijgt en de grootste oplevert

Roep de functie aan met getallen (4,5) en druk de grootste af.

```
#FUNCTIE DEFINITIE
#deze functie levert de grootste op
def                ( getal1, getal2 ):
    if
    else:
```

Functie grootste: stap 2

Schrijf nu zelf een functie die twee getallen als parameters binnen krijgt en de grootste oplevert

Roep de functie aan met getallen (4,5) en druk de grootste af.

```
#FUNCTIE DEFINITIE
#deze functie levert de grootste op
def isGrootste ( getal1, getal2 ):
    if getal1 >= getal2:
        return getal1
    else:
        return getal2

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
                        .
print( "De grootste is",                    )
```

Scope (oftewel: levensduur)

```
#FUNCTIE DEFINITIE
#deze functie levert de grootste op
def isGrootste ( getal1, getal2 ):
    if getal1 >= getal2:
      return getal1
    else:
      return getal2

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
grootsteGetal = isGrootste( 4, 5)
print( "De grootste is", grootsteGetal )
```

In onze programma krijgt `getal1` de waarde 4.

1) Wanneer wordt `getal1` gecreëerd?

2) Wat gebeurt er met `getal1` **na** `grootsteGetal`?

Lokale variabele: bestaat alleen binnen een functie

Globale variabele

```
minimum = 1
#FUNCTIE DEFINITIE
#deze functie levert de grootste op
def isGrootste ( getal1, getal2 ):
    if getal1 >= getal2:
      return getal1
    else:
      return getal2

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
grootsteGetal = isGrootste( 4, 5)
print("Grootste:", grootsteGetal,"en kleinste:", minimum)
```

En als je een variabele buiten een functie wilt gebruiken?

- Bovenaan definiëren en een waarde geven
- Levensduur: hele programma

Globale variabele: bestaat gedurende hele programma

Functie aanroep met parameters (in plaats van ‘harde’ waardes)

```
#VARIABELEN EN CONSTANTEN
lengte_eerste_zijde = 5
lengte_tweede_zijde = 3

#FUNCTIE DEFINITIE
#deze functie levert de omtrek van een rechthoek op
def omtrekRechthoek ( breedte, hoogte ):
    omtrek = 2*breedte + 2*hoogte
    return omtrek

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
omtrek = omtrekRechthoek( lengte_eerste_zijde, lengte_tweede_zijde )
```

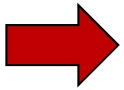
Functie aanroep met parameters (in plaats van ‘harde’ waardes)

```
#VARIABELEN EN CONSTANTEN
lengte_eerste_zijde = 5
lengte_tweede_zijde = 3

#FUNCTIE DEFINITIE
#deze functie levert de omtrek van een rechthoek op
def omtrekRechthoek ( breedte, hoogte ):
    omtrek = 2*breedte + 2*hoogte
    return omtrek

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
omtrek = omtrekRechthoek( lengte_eerste_zijde, lengte_tweede_zijde )
```

Afhandelen foutieve invoer

- Wat doet de functie bij omtrekRechthoek(-5, 3)?
- Breedte en hoogte moeten natuurlijk > 0 zijn
- Een functie met een return moet **altijd iets** opleveren

- Afspraak: lever iets ongebruikelijks op, bijvoorbeeld -1



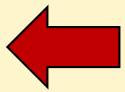
```
def omtrekRechthoek ( breedte, hoogte ):
    if breedte <= 0 or hoogte <= 0:
        return -1
    else:
        omtrek = 2*breedte + 2*hoogte
        return omtrek
```

Fout afhandelen buiten functie

```
#VARIABELEN EN CONSTANTEN
lengte_eerste_zijde = 5
lengte_tweede_zijde = 3

#FUNCTIE DEFINITIE
#deze functie levert de omtrek van een rechthoek op
#levert -1 op bij ongeldige invoer (parameter waarden 0 of kleiner)
def omtrekRechthoek ( breedte, hoogte ):
    if breedte <= 0 or hoogte <= 0:
        return -1
    else:
        omtrek = 2*breedte + 2*hoogte
        return omtrek

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
omtrek = omtrekRechthoek( lengte_eerste_zijde, lengte_tweede_zijde )

#controleren op geldigen invoer en antwoord afdrukken
if omtrek == -1: #er is ongeldige invoer gegeven
    print( "De getallen kunnen niet worden gebruikt." )
else:
    print( "De omtrek van je rechthoek is", omtrek )
```

Fout afhandelen buiten functie

```
#FUNCTIE DEFINITIE
#deze functie levert de omtrek van een rechthoek op
#levert -1 op bij ongeldige invoer (parameter waarden 0 of kleine
def omtrekRechthoek ( breedte, hoogte ):
    if breedte <= 0 or hoogte <= 0:
        return -1
    else:
        omtrek = 2*breedte + 2*hoogte
        return omtrek

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
omtrek = omtrekRechthoek( 5, 3 )

#controleren op geldigen invoer en antwoord afdrukken
if omtrek == -1: #er is ongeldige invoer gegeven
    print( "De getallen kunnen niet worden gebruikt." )
else:
    print( "De omtrek van je rechthoek is", omtrek )
```



Programma structuur:

```
#VARIABELEN EN CONSTANTEN
PI = 3.14

#FUNCTIE DEFINITIE
def printHallo()
    //acties of berekeningen:
    print( “Hello World!” )

#HOOFDPROGRAMMA MET AANROEP VAN FUNCTIE
printHallo()
```

Variabelen **definiëren**

Functie **definitie**

Functie **aanroep**

Bij grote documenten is het lastig dingen terug te vinden.

Afspraak: Boven aan je bestand zet je:

- Variabelen en constanten
- Functie definities

Verwerkings opdrachten

- Hoofdstuk 8 Eigen Functies: Theorie en opgaven
  - Maak alle opdrachten
  - (Optioneel mag je overslaan)
  - (Afsluitende opgaven zijn voor volgende week)
- Turtle Graphics - Functies en Parameters: opgaven
  - Niet te lang aan 1 opgave blijven werken
  - Hoeven niet allemaal af

Tot slot

- Bij programmeren maak je gebruik van **functies**.
- Een functie heeft een **naam** en soms één of meerdere **parameters**.
- De **functiedefinitie** beschrijft wat de functie doet.
- Als je de functie wilt gebruiken, doe je een **functieaanroep.**
- Gebruik **commentaar, zinvolle namen,** en **vaste structuur**
- Je programma wordt overzichtelijker en kun je onderdelen makkelijker vaker gebruiken.

Mobiele internetbundel

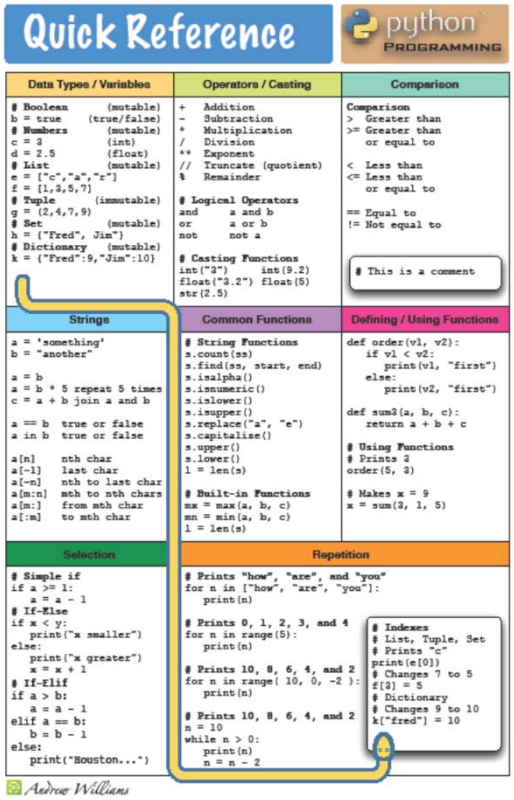
Bij Simyo: € 1 = 30 MB

Maak een website die uitrekent:

a) Als je Euro's invult: hoeveel MB je krijgt.

b) Als je MBs invult: hoeveel je moet betalen.

SpiekBriefje



Quick Reference

python PROGRAMMING

Data Types / Variables

```
# Boolean       (mutable)
b = true     (true/false)
# Numbers       (mutable)
c = 3             (int)
d = 2.5           (float)
# List          (mutable)
e = ["c","a","r"]
f = [1,3,5,7]
# Tuple        (immutable)
g = (2,4,7,9)
# Set           (mutable)
h = {"Fred", Jim"}
# Dictionary    (mutable)
k = {"Fred":9,"Jim":10}
```

Operators / Casting

```
+   Addition
-   Subtraction
*   Multiplication
/   Division
**  Exponent
//  Truncate (quotient)
%   Remainder

# Logical Operators
and      a and b
or       a or b
not      not a

# Casting Functions
int("2")      int(9.2)
float("3.2") float(5)
str(2.5)
```

Comparison

```
Comparison
>  Greater than
>= Greater than
   or equal to

<  Less than
<= Less than
   or equal to

== Equal to
!= Not equal to

# This is a comment
```

Strings

```
a = 'something'
b = "another"

a = b
a = b * 5 repeat 5 times
c = a + b join a and b

a == b  true or false
a in b  true or false

a[n]     nth char
a[-1]    last char
a[-n]    nth to last char
a[m:n]   mth to nth chars
a[m:]    from mth char
a[:m]    to mth char
```

Common Functions

```
# String Functions
s.count(ss)
s.find(ss, start, end)
s.isalpha()
s.isnumeric()
s.islower()
s.isupper()
s.replace("a", "e")
s.capitalize()
s.upper()
s.lower()
l = len(s)

# Built-in Functions
mx = max(a, b, c)
mn = min(a, b, c)
l = len(s)
```

Defining / Using Functions

```
def order(v1, v2):
    if v1 < v2:
        print(v1, "first")
    else:
        print(v2, "first")

def sum3(a, b, c):
    return a + b + c

# Using Functions
# Prints 2
order(5, 2)

# Makes x = 9
x = sum(2, 1, 5)
```

Selection

```
# Simple if
if a >= 1:
    a = a - 1
# If-Else
if x < y:
    print("x smaller")
else:
    print("x greater")
    x = x + 1
# If-Elif
if a > b:
    a = a - 1
elif a == b:
    b = b - 1
else:
    print("Houston...")
```

Repetition

```
# Prints "how", "are", and "you"
for n in ["how", "are", "you"]:
    print(n)

# Prints 0, 1, 2, 3, and 4
for n in range(5):
    print(n)

# Prints 10, 8, 6, 4, and 2
for n in range( 10, 0, -2 ):
    print(n)

# Prints 10, 8, 6, 4, and 2
n = 10
while n > 0:
    print(n)
    n = n - 2
```

```
# Indexes
# List, Tuple, Set
# Prints "c"
print(e[0])
# Changes 7 to 5
f[3] = 5
# Dictionary
# Changes 9 to 10
k["fred"] = 10
```

Andrew Williams

Fouten opsporen

- Kijk naar grensgevallen, vreemde waarden, belangrijke groepen van waarden
  - waarde 0
  - voor negatieve getallen
  - enz.

- Debuggen:
  - met trace-optie
  - tracing table

Belangrijke aspecten bij programmeren

- **Volledigheid**: Heeft het eindproduct alles wat verwacht wordt?
- **Correctheid**: Doet alles het zoals je zou verwachten?
- **Constructie**: Hoe goed is de code geschreven?
  - duidelijk en logisch opgebouwd
  - gebruikt loops condities en functies
  - 'harde' waardes worden vermeden
- **Stijl**: Is de code leesbaar?
  - Zinvolle commentaar
  - LogischeNamen (in camelCase)

Goed programmeren gaat dus **niet alleen** om een werkende **oplossing**, maar juist om een **degelijke aanpak en oplossing**.