

Flowcharts en Functies



BIN-OWE1

Studiewijzer

Les	Onderwerp	
	Algemeen	Python
1	Linux	
2	Git	Input/output
3	Pseudocode	If/elif/else Booleans
4		For loop
5		Lists and tuples Files (CSV bestanden)
6	Flowchart	Funcities
7		Strings CSV bestanden

Studiemateriaal

- Boek: “Starting Out with Python, third edition”
- Reader: “Linux voor Bio-informatici”
- Onderwijs Online
- Kom je er niet uit? → Google



Doelstellingen

- Aan het eind van deze week kan je
 - Pseudocode omzetten naar een flowchart
 - Een flowchart gebruiken om je code te schrijven
 - Functies toepassen in Python
- Aan het eind van deze week begrijp je
 - De verschillende onderdelen van een flowchart
 - Hoe functies werken
 - Wat een parameter, argument en return statement zijn

Inhoud

- **Introductie**
- Ontwerp en aanroep van functies
- Ontwerp van een programma
- Lokale variabelen
- Argumenten
- Globale variabelen
- Returns

Wat zijn functies?

- Een afgebakend stukje code dat bestaat uit bij elkaar horende statements
- Vergelijkbaar met een wiskundige functie: je stopt er een waarde in, er vindt een bewerking plaats en er volgt een uitkomst.
- We hebben er stiekem al best een paar gezien: `print()`, `input()`, `upper()`, `replace()` etc.

Wat zijn de voordelen?

- Code wordt eenvoudiger
- Hergebruik van code
- Betere mogelijkheden om te debuggen
- Sneller ontwikkelen van code
- Eenvoudiger om in teams te werken

Syntax van een functie

```
def <naam>() :
```

```
    statement
```

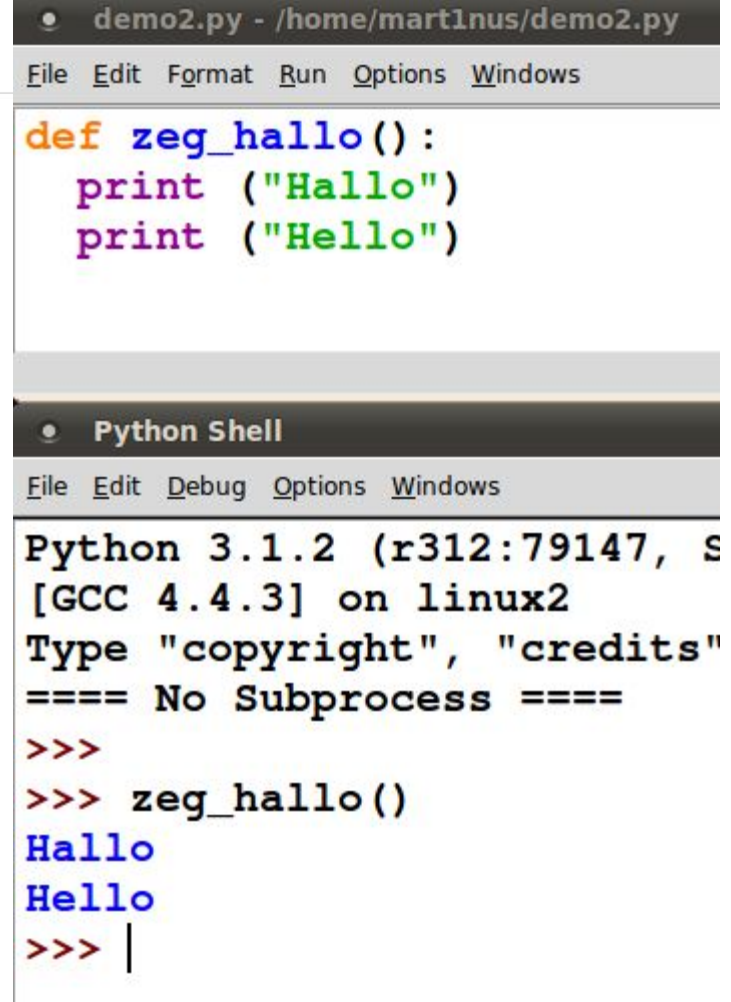
```
    statement
```

```
    statement
```

```
    statement
```


Voorbeeld

```
def zeg_hallo():  
    print("Hello")  
    print("Hallo")
```



The screenshot shows a Python IDE with two windows. The top window, titled 'demo2.py - /home/mart1nus/demo2.py', contains a Python function definition: `def zeg_hallo():` followed by two indented `print` statements: `print("Hollo")` and `print("Hello")`. The bottom window, titled 'Python Shell', shows the output of running the script. It displays the Python version (3.1.2), GCC version (4.4.3), and the system (linux2). It also shows the command `zcg_hallo()` being executed, which results in the output `Hollo` and `Hello` on separate lines.

```
demo2.py - /home/mart1nus/demo2.py  
File Edit Format Run Options Windows  
  
def zeg_hallo():  
    print("Hollo")  
    print("Hello")  
  
Python Shell  
File Edit Debug Options Windows  
  
Python 3.1.2 (r312:79147, S  
[GCC 4.4.3] on linux2  
Type "copyright", "credits"  
==== No Subprocess ====  
  
>>>  
>>> zcg_hallo()  
Hollo  
Hello  
>>> |
```

Inhoud

- Introductie
- **Ontwerp en aanroep van functies**
- Ontwerp van een programma
- Lokale variabelen
- Argumenten
- Globale variabelen
- Returns

Functie Flow

1. Run het programma
2. Python lees het hele programma in
3. Ziet de aanroep van `main()`
4. Weet deze te vinden en voert `main()` uit
5. Ziet de aanroep van `zeg_hallo()`
6. Weet deze te vinden en voert `zeg_hallo` uit()
7. Einde programma

```
demo2.py - /home/martinus/demo2.py
File Edit Format Run Options Windows

def main():
    zeg_hallo()

def zeg_hallo():
    print ("Hallo")
    print ("Hello")

main()

Python Shell
File Edit Debug Options Windows

Python 3.1.2 (r312:79147, Sep 17 2008, 16:11:14)
[GCC 4.4.3] on linux2
Type "copyright", "credits" or "help()" to get more help.
>>>
Hallo
Hello
>>>
```

Inhoud

- Introductie
- Ontwerp en aanroep van functies
- **Ontwerp van een programma**
- Lokale variabelen
- Argumenten
- Globale variabelen
- Returns

Flowcharts en Functies

```

demo2.py - /home/martlnus/demo2.py
File Edit Format Run Options Windows

def main():
    zeg_hallo()

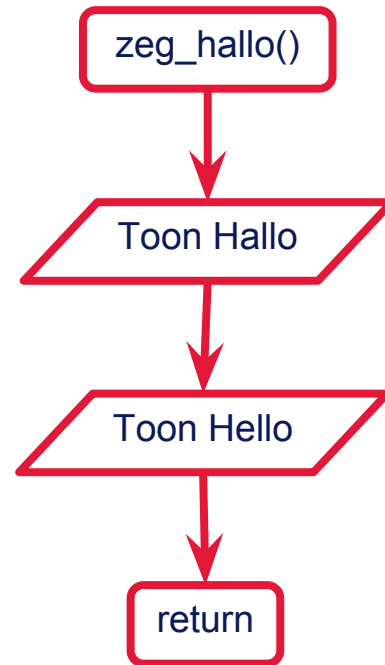
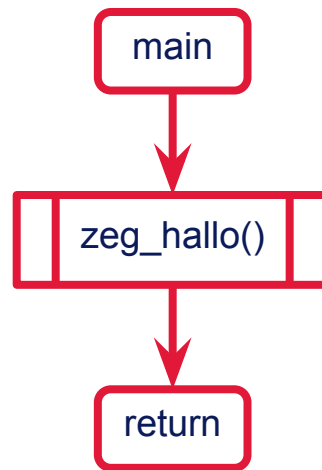
def zeg_hallo():
    print ("Hallo")
    print ("Hello")

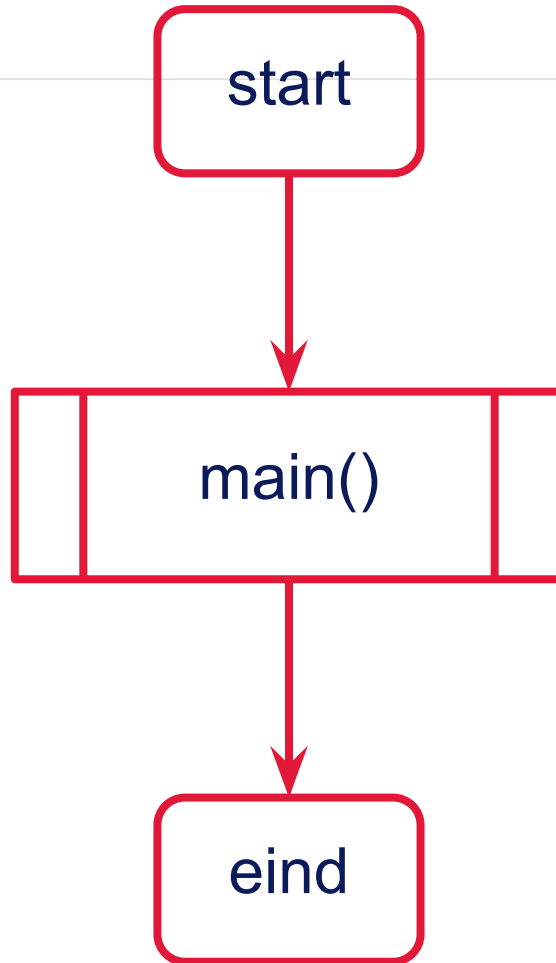
main()
    
```

```

Python Shell
File Edit Debug Options Windows

Python 3.1.2 (r312:79147, Sep 17 2008, [GCC 4.4.3] on linux2
Type "copyright", "credits" or "help()" to get more help.
>>>
Hallo
Hello
>>>
    
```





Inhoud

- Introductie
- Ontwerp en aanroep van functies
- Ontwerp van een programma
- Lokale variabelen
- Argumenten
- Globale variabelen
- Returns

Herhaling Variabelen

- Een <naam> koppelen aan een <waarde>

```
<naam> = <waarde>
```

```
var1 = "a"
```

- Variabelen zijn overschrijfbaar waardoor <naam> een andere <waarde> krijgt

```
var1 = "a"
```

```
var1 = "b"
```

```
print(var1)
```

```
>>> 'b'
```


Herhaling Variabelen

- De <waarde> is doorgeefbaar naar andere variabelen via de naam van de variabele

```
<naam1> = <waarde>
```

```
<naam2> = <naam1>
```

```
var1 = "a"
```

```
var2 = var1
```

```
print(var2)
```

```
>>> 'a'
```

Lokale Variabelen

- **Lokale variabelen** zijn variabelen die alleen binnen de functie bekend zijn.
- Dit heeft te maken met de **scope** (invloedsgebied) van de functie.
- Dit houdt in dat de functie niets over de buitenwereld weet...
- ...maar de buitenwereld dus ook niets over de functie (black box)

Juist Voorbeeld

```
def main():  
    zeg_hallo()
```

```
def zeg_hallo():  
    naam = input ("Wat is je naam")  
    print ("Hallo ", naam)
```

```
main()
```

Kan iemand de flow nogmaals toelichten?

Onjuist Voorbeeld

```
def main():  
    zeg_hallo()  
    print ("Hallo ", naam)  
def zeg_hallo():  
    naam = input ("Wat is je naam: ")
```

```
main()
```

```
>>>
```

```
Wat is je naam: Martijn
```

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

```
  File "/home/martinius/demo2.py", line 8, in <module>
```

```
    main()
```

```
  File "/home/martinius/demo2.py", line 3, in main
```

```
    print ("Hallo ", naam)
```

```
NameError: global name 'naam' is not defined
```

```
>>>
```

De functie `main()` kan dus de variabele `naam` niet zien omdat dit een lokale variabele is.

Samenvatting

- Functies zijn een afgebakend stukje code, een soort black box
- Functies maken je code makkelijker om te hergebruiken
- Functies kennen lokale variabelen, die dus niet te benaderen zijn buiten de functie om
- Functies mogen elkaar aanroepen, bijvoorbeeld de `main()` functie die een andere functie aanroept, waardoor ze een soort doorgeefluik worden

Inhoud

- Introductie
- Ontwerp en aanroep van functies
- Ontwerp van een programma
- Lokale variabelen
- **Argumenten**
- Globale variabelen
- Returns

Parameters

- Hoewel ik van buitenaf geen invloed kan uitoefenen op hoe een functie werkt, mag ik wel input meegeven
- In de aanroep noem je dit **argumenten**
- In de definitie noem je dit **parameters**
- Vaak worden deze termen door elkaar gebruikt
- We hebben wel eens vaker gebruik gemaakt van argumenten, in functies als `print("Hello")` en `seq.replace("^", "")`

Syntax

```
def <naam>(par1) :  
    statement (doe iets met par1)
```


Parameter of Argument?

```
def naam_print(naam):  
    print(naam)
```

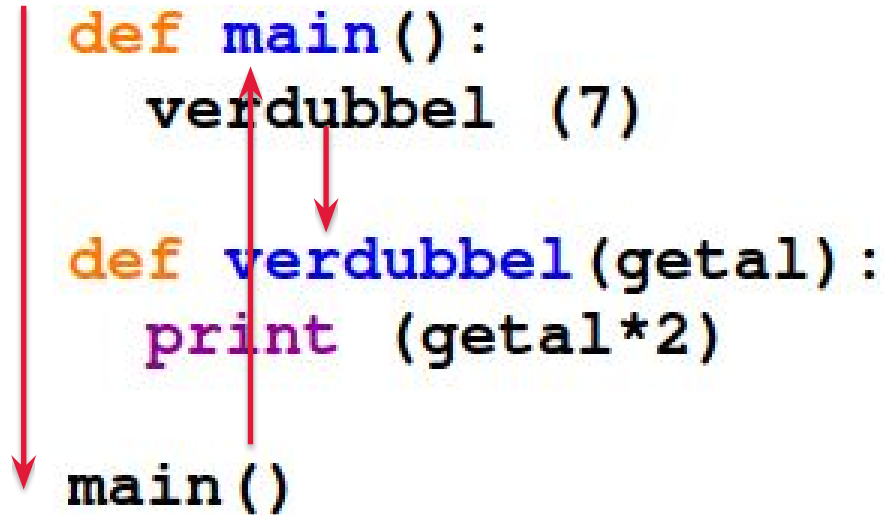
Welke is naam?

```
naam_print("Esther")
```

Welke is "Esther"?

Voorbeeld Functie

```
def main():  
    verdubbel (7)  
  
def verdubbel(getal):  
    print (getal*2)  
  
main()
```



Wat doet deze functie?

Voorbeeld Functie

```
def main():  
    verdubbel (7)
```

```
def verdubbel(getal):  
    print (getal*2) 14
```

```
main()
```

De functie `verdubbel()` krijgt een waarde mee (7)

Deze wordt overgedragen aan de parameter `getal` en gebruikt in de functie `(getal*2)`

Meerdere argumenten

- Een functie kan meerdere argumenten accepteren, net als `seq.replace("^", "")` dat kan
- Hierbij was dus de originele syntax met bijhorende parameters `<str>.replace(old, new)`

Meerdere argumenten

```
def main():  
    vermenigvuldig (7,2)  
  
def vermenigvuldig(getal1, getal2):  
    print (getal1*getal2)  
  
main()
```

Vraag

- Wat is de uitkomst van dit programma?

```
def main():  
    verminder(5, 7)  
  
def verminder(a, b):  
    print(a-b)  
  
main()
```

Defaults en Optionele Parameters

- Default betekent standaard
- De default uitslag van een voetbalwedstrijd is 0-0, zonder verdere actie is dit ook de eindstand
- We kunnen een functie ook **defaults** meegeven als **parameter**

```
def vermenigvuldig(x=2, y=4):  
    print(x*y)
```

- Dit zorgt ervoor dat de **parameter** ook **optioneel** wordt, je hoeft hem niet perse in te vullen om een uitkomst te krijgen

```
def main():  
    vermenigvuldig (7,2)  
  
def vermenigvuldig(getal1=1, getal2=1):  
    print (getal1*getal2)  
  
main()
```


Vraag

- Wat is de uitkomst van dit programma?

```
def main():  
    verminder(5, 7)  
  
def verminder(a=4, b=3):  
    print(a-b)  
  
main()
```

Aanroep

- Welk van deze aanroepen is correct voor de functie `vermenigvuldig()`

```
def vermenigvuldig(x=2, y=4):  
    print(x*y)
```

1. `vermenigvuldig()`
2. `vermenigvuldig(7,2)`
3. `vermenigvuldig(7)`

Pass by position

- Overdracht van waarde op basis van de positie in de argumentenlijst

```
def main():  
    vermenigvuldig (7,2)  
  
def vermenigvuldig(getal1=1, getal2=1):  
    print (getal1*getal2)  
  
main()
```

Pass by name

- Overdracht van waarde op basis van de naam van de parameter

```
def main():  
    vermenigvuldig (getal2=7)  
  
def vermenigvuldig(getal1=1, getal2=1):  
    print (getal1*getal2)  
  
main()
```

Vraag

- Wat is de uitkomst van dit programma?

```
def main():  
    verminder(7)
```

```
def verminder (a=4, b=3):  
    print (a-b)
```

```
main()
```

Vraag

- Wat is de uitkomst van dit programma?

```
def main():  
    verminder(b=7, a=2)  
  
def verminder(a=4, b=3):  
    print(a-b)  
  
main()
```

Samenvatting

- Argumenten zijn waarden die je aan een functie kan geven als een vorm van input
- Deze waarde is gekoppeld aan de parameter van de functie
- Parameters kunnen een default waarde meekrijgen in de definitie, waardoor ze optioneel worden
- De parameters kunnen waarde krijgen door de argumenten op basis van positie of naam te vullen

Inhoud

- Introductie
- Ontwerp en aanroep van functies
- Ontwerp van een programma
- Lokale variabelen
- Argumenten
- **Globale variabelen**
- Returns

Globale variabelen

- Een **globale variabele** is het tegenovergestelde van een **lokale variabele**. De **scope** van de globale variabele is het gehele programma.
- Met het sleutelwoord **global** refereer je aan eerder gedeclareerde variabelen

```
getal = 7
```

→ Dit is de globale variabele

```
def main():
```

```
    global getal
```

```
    print (getal)
```

Getal is niet deel van een functie maar deel van het gehele programma.

→ Binnen een functie maak je een onderscheid door het sleutelwoord **global** toe te voegen.

```
main()
```

```
|
```

Inhoud

- Introductie
- Ontwerp en aanroep van functies
- Ontwerp van een programma
- Lokale variabelen
- Argumenten
- Globale variabelen
- Returns

Returns

- Naast dat het handig is als ik een functie waardes mee mag geven, is het ook handig als een functie iets oplevert
- Tot nu toe alleen print statements gezien als output
- Een functie kan ook iets teruggeven aan de buitenwereld met behulp van een `return`

Return Statement

```
def <naam>(pars):  
    statement  
    statement  
    return statement
```

Return Statement

```
def vermenigvuldig(getal1, getal2):  
    som = getal1 * getal2  
    return som
```

Returns

- Zonder return levert een functie `None` op, iets leegs
- Je kan er ook bewust voor kiezen om een `None` te returnen, door een lege return te doen
- De `return` beëindigd de functie ook direct

Return None

```
def vermenigvuldig(getal1, getal2):  
    som = getal1 * getal2  
    return
```

```
print(type(vermenigvuldig(4,5)))  
>>> <class 'NoneType'>
```


Returns

- Ik mag meer dan 1 variabele retourneren met behulp van `return`, dit levert dan een tuple op

Herhaling Tuples

- Een onwijzigbaar koppel maken van waardes

```
<tup> = (<item1>, item2, ...)
```

```
var1 = "a"
```

```
var2 = 1
```

```
tup1 = (var1, var2)
```

```
print(tup1)
```

```
>>> ('a', 1)
```

Herhaling Tuples

- Een tuple heeft indices

```
tup1 = ("a","b")  
print(tup1[0])  
>>> 'a'
```

Multiple Return

```
def vermenigvuldig(getal1, getal2):  
    som = getal1 * getal2  
    return som, getal1  
  
print(vermenigvuldig(4,5))  
>>> (20,4)
```

Uitkomst Multiple Return

- Ik kan twee dingen doen met de uitkomst van de multiple return:
 - Opslaan in 1 variabele
 - Dan blijft het een tuple
 - Opslaan in meerdere variabelen
 - Dan splits ik dus de tuple op in losse variabelen van het originele data type

Return naar 1 variabele

```
def vermenigvuldig(getal1, getal2):  
    som = getal1 * getal2  
    return som, getal1
```

```
uitkomst = vermenigvuldig(4,5)  
print(type(uitkomst))  
>>> <class 'tuple'>
```

Return naar meerdere variabelen

```
def vermenigvuldig(getal1, getal2):  
    som = getal1 * getal2  
    return som, getal1
```

```
uitkomst, getal1 = vermenigvuldig(4,5)  
print(type(uitkomst))  
>>> <class 'int'>
```

Samenvatting

- Het return statement beëindigd de functie direct
- Ik kan een functie meerdere dingen laten teruggeven:
 - None
 - 1 variabele
 - Meerdere variabelen

Opdracht

- Ga naar OnderwijsOnline
- Maak Afvinkopdracht 6
- Bij problemen:
 - Google
 - Klasgenoten
 - Docent
- Lever deze in bij de praktijkdocent





Nog meer vragen!

Vraag

- Wat is de uitkomst van dit programma?

```
def main():  
    x = 2  
    verminder(b=7, a=2)  
  
def verminder(a=4, b=3):  
    print(a-b*x)  
  
main()
```

Vraag

- Wat is de uitkomst van dit programma?

```
x = 2
def main():
    verminder(b=7, a=2)

def verminder (a=4, b=3):
    global x
    print (a-b*x)

main()
```

```
x = 7
```

Vraag

- Wat is de uitkomst van dit programma?

```
def main():  
    global x  
    doeIets(x)  
    doeIetsAnders(d=x, c=5)  
    print(x)
```

```
def doeIets(a=5, b=2):  
    global x  
    x = a*b
```

```
def doeIetsAnders(c=0, d=0):  
    global x  
    x = c-d  
    doeIets(x, x)
```

```
main()
```

```
x = 7
```

Vraag

- Wat is de uitkomst van dit programma?

```
def main():  
    global x  
    doeIets(x)  
    doeIetsAnders(d=x, c=5)  
    print(x)
```

```
def doeIets(a=5, b=2):  
    #global x  
    x = a*b
```

```
def doeIetsAnders(c=0, d=0):  
    global x  
    x = c-d  
    doeIets(x, x)
```

```
main()
```

Verantwoording

- In deze uitgave is géén auteursrechtelijk beschermd werk opgenomen
- Alle teksten © Martijn van der Bruggen/Esther Kok/HAN tenzij expliciet externe bronnen zijn aangegeven
- Screenshots op basis van eigen werk auteur en/of vernoemde sites en/of fair use
- Eventuele images zijn opgenomen met vermelding van bron