

# Algoritme van Euclides (300 v. Chr.)

Algoritmes, dat is, beschrijvingen van systematische manieren om een concreet probleem op te lossen, bestaan al eeuwenlang. Denk bijvoorbeeld aan de technieken die je leerde om getallen neergeschreven in het tientallig stelsel op te tellen, te vermenigvuldigen, of te delen. De technieken die je leerde bestaan uit een aantal vastgelegde stappen die in de juiste volgorde uitgevoerd moeten worden en voor elke mogelijke invoer werken.

In deze notebook zullen we het [algoritme van Euclides \(https://nl.wikipedia.org/wiki/Algoritme\\_van\\_Euclides\)](https://nl.wikipedia.org/wiki/Algoritme_van_Euclides) dat de grootste gemene deler (GGD) van twee getallen bepaalt, bekijken en implementeren in Python.

**In deze notebook is het belangrijk dat je de stappen die worden gemaakt, begrijpt en de code die wordt ontwikkeld, kan lezen. De exacte syntax van de commando's (=correcte manier om commando's in Python te gebruiken) komt in volgende notebooks nog uitgebreid aan bod. Het leerdoel van deze notebook is dus om de basisconstructies van Python op een abstracte manier te leren kennen en inzicht te verwerven in hoe we een probleem omzetten in code.**

## Beschrijving van het algoritme van Euclides

Het algoritme van Euclides werkt als volgt:

Probleem: Bepaal GGD van 2 getallen Invoer: 2 getallen Uitvoer: de GGD van deze twee getallen 1. Noem het grootste van de beide getallen A, het andere B. 2. Trek B net zo vaak van A af totdat er 0 overblijft of een getal kleiner dan B ( $A \bmod B$ ). 3. Wanneer er 0 overblijft, is B de ggd. 4. Zo niet, herhaal dan het algoritme met B en wat er van A over is.

We illustreren het algoritme met een voorbeeld:

Bereken de GGD van 35 en 49 1.  $A=49$ ,  $B=35$  2. Trek B af van A:  $A=49-35=14$  A (14) is kleiner dan B (35) 3. A is niet 0, we moeten dus voort naar stap 4 4. We moeten het algoritme herhalen voor 14 en 35 Bereken de GGD van 14 en 35 1.  $A=35$ ,  $B=14$  2. Trek B af van A:  $A=35-14=21$  A (21) is niet kleiner dan B (14) Trek B af van A:  $A=21-14=7$  A (7) is nu kleiner dan B (14) 3. A is niet 0, we moeten dus voort naar stap 4 4. We moeten het algoritme herhalen voor 7 en 14 Bereken de GGD van 7 en 14 1.  $A=14$ ,  $B=7$  2. Trek B af van A:  $A=14-7=7$  A (7) is niet kleiner dan B (7) Trek B af van A:  $A=7-7=0$  A (0) is nu kleiner dan B (7) 3. A is 0, dus we kunnen stoppen en de GGD is 7

Probeer zelf even een paar voorbeeldjes uit om te testen dat je het algoritme helemaal correct begrepen hebt. Gebruik het algoritme van Euclides om volgende grootste gemene delers na te rekenen:

- $\text{GGD}(64,16) = 16$
- $\text{GGD}(23,11) = 1$
- $\text{GGD}(51,119) = 17$

## Implementatie van het algoritme van Euclides in Python

We moeten het algoritme nu gaan vertalen in stappen die de *Python interpreter* begrijpt. In grote lijnen gaan we gebruik maken van volgende constructies:

- *Invoer en uitvoer*
- *Variabelen*: Bewaren van een waarde om later te gebruiken
- *Conditionele uitvoering*: Indien ... dan ... anders ...
- *Herhalingslus*: Zolang ... doe ...

## Invoer en uitvoer

Voorlopig geven we de invoer van het programma in het programma zelf; dit noemen we *hard coding* van de input. Later zullen we zien hoe we invoer rechtstreeks aan de gebruikers kunnen vragen. We gebruiken de *variabelen* *X* en *Y* om de invoer van ons programma op te slaan; dat is: we geven *X* en *Y* via volgende commando's de waarden waarvan we de grootste gemene deler willen berekenen:

In [1]:

```
X=49
Y=35
```

Variabelen zijn grootheden die we zelf definiëren en waarmee we kunnen rekenen:

In [2]:

```
2*X+Y
```

Out[2]:

133

We kunnen het resultaat van een berekening aan een andere variabele toekennen, of zelfs terug aan een van de variabelen die we reeds definieerden:

In [3]:

```
X=X-Y
```

In een notebook kan je het resultaat steeds opvragen door gewoon de variabele naam neer te schrijven:

In [4]:

```
X
```

Out[4]:

14

In een programma kunnen we ook de *functie* *print(.)* gebruiken:

In [5]:

```
X=49
Y=35

print(X) # Geef inhoud van X
print(Y) # Geef inhoud van Y
print(X,Y) # print zowel X als Y af
print(3*X,X-Y) # print het resultaat van de twee bewerkingen
               # deze bewerkingen passen het resultaat van X en Y niet aan
```

```
49
35
49 35
147 14
```

De eerste twee regels van ons programma zijn dus als volgt:

In [6]:

```
X=49
Y=35
```

De Python interpreter zal deze regels na elkaar uitvoeren. Willen we later de grootste gemene deler van 2 andere getallen berekenen, dan moeten we enkel deze twee regels aanpassen om de nieuwe input te geven.

## Stap 1: Grootste van 2 getallen

De eerste stap is de volgende:

1. Noem het grootste van de beide getallen A, het andere B

Uiteraard zouden we dit manueel kunnen doen; aangezien we de invoerwaarden  $X$  en  $Y$  hargecodeerd hebben, zouden we bij conventie kunnen voorstellen dat  $X$  de grootste en  $Y$  de kleinste waarde moet bevatten. We willen echter algemene code schrijven die niet van zulke aannames moet uitgaan. Later gaan we  $X$  en  $Y$  rechtstreeks aan de gebruiker van ons programma vragen en dan moeten we uiteraard er wel rekening mee houden dat de gebruiker mogelijk niet op de hoogte is van deze conventie, of zelfs bewust weigert hiermee rekening te houden.

Afhankelijk of de *conditie*  $X < Y$  waar of vals is, moeten we dus ofwel  $A=Y$  en  $B=X$  uitvoeren of  $A=X$  en  $B=Y$  zodat  $A$  de grootste waarde van de twee bevat. Dit noemen we *conditionele uitvoering* en hiervoor zullen we de *if-constructie* moeten gebruiken. Die werkt als volgt:

In [7]:

```
X=49
Y=35

if X<Y:      # Als X<Y
    A=Y      # Voer dan deze twee
    B=X      # regels uit
else:        # Anders
    A=X      # deze twee
    B=Y      # regels

print(A,B)
```

49 35

Voer de code uit. Pas  $X$  en  $Y$  aan en controleer dat  $A$  en  $B$  de juiste waardes krijgen. Bij het schrijven van een *if* moet je letten op volgende syntax-vereisten die door een beginnende programmeur vaak over het hoofd worden gezien:

- *if* en *else* hebben geen hoofdletter;
- vergeet niet om een dubbele punt (:) te schrijven na de conditie van de *if* en na *else*;
- de regels die conditioneel worden uitgevoerd, moeten inspringen. Gebruik hiervoor 4 spaties. Vermijd het mengen van spaties en tabs. Een goede Python editor zal veel van de indentatie voor z'n rekening nemen;
- de indentatie van *else* moet terug hetzelfde zijn als die van *if*;
- door de code terug te laten inspringen op het niveau van *if* en *else* duid je aan dat het conditionele blok afgelopen is. Dat is: in de code hierboven staat *print* terug op dezelfde hoogte als *if* en *else*. Daaruit leidt de Python interpreter af dat *print(A,B)* niet meer tot de *else*-tak van de conditionele uitvoering behoort.

Heb je de code niet helemaal door? Speel dan even met [deze code in de handige tool Python Tutor](http://pythontutor.com/iframe-embed.html#code=%23%20Invoer%0AX%20%3D%2049%0AY%20%3D%2035%0A%0Aif%20X%3EY%3A%08print%28A%2C%20B%29%0A%0A#textReferences=false) (<http://pythontutor.com/iframe-embed.html#code=%23%20Invoer%0AX%20%3D%2049%0AY%20%3D%2035%0A%0Aif%20X%3EY%3A%08print%28A%2C%20B%29%0A%0A#textReferences=false>) waarbij je stap per stap door de code kan lopen en bekijken wat er gebeurt.

## Stap 2: Bereken A modulo B

De tweede stap van het algoritme is de volgende:

1. Trek  $B$  net zo vaak van  $A$  af totdat er 0 overblijft of een getal kleiner dan  $B$  ( $A \bmod B$ ).

Merk op dat je op deze manier de rest van de deling  $A/B$  berekent; als  $A=kB+r$ , met  $r<B$ , dan zal je  $k$  maal  $B$  van  $A$  kunnen aftrekken en daarna blijft  $r$  over.

Om deze stap te implementeren zullen we gebruik maken van een *herhalingslus*. Dit is een groep commando's die herhaald worden tot er aan een bepaalde conditie voldaan is. In dit geval moet het volgende gebeuren:

Zolang  $B \leq A$ , herhaal:  $A = A - B$

In Python doen we dat met behulp van een *while-loop*:

In [8]:

```
while B<=A:  # Zolang B<A
    A = A - B      # Herhaal: A wordt A-B
```

Opnieuw moeten we erop letten dat:

- *while* zonder hoofdletter wordt geschreven;
- na de conditie een dubbele punt (:) volgt;
- de regel(s) die herhaald moeten worden inspringen;
- de eerste regel die niet meer herhaald moet worden, springt terug in.

Test de code uit met verschillende waarden voor *A* en *B*. Print het resultaat na de *while*-loop af.

## Stappen 3 en 4 : Herhaal tot *A=0*

Stappen 3 en 4 zijn nu als volgt:

1. Wanneer er 0 overblijft, is *B* de ggd.
2. Zo niet, herhaal dan het algoritme met *B* en wat er van *A* over is.

Merk op dat hoewel dit is neergeschreven alsof het een opeenvolging van stappen betreft, dit eigenlijk volgende herhaling inhoudt. Merk op dat de rol van *A* en *B* net iets anders is dan in het algoritme hierboven, omdat we *A* en *B* steeds omwisselen nadat we de herhalingslus van stap 2 hebben uitgevoerd:

Zolang *B* niet 0 is, herhaal: Voer de herhalingslus van stap 2 uit Wissel *A* en *B* om (zodat *A* terug de grootste is)  
Het antwoord is *A*

Vergewis jezelf ervan dat deze stappen inderdaad correct zijn door het voorbeeld van het begin van deze notebook nog eens grondig te bekijken.

## Omwisselen van *A* en *B*

We bekijken nu eerst de code om *A* en *B* om te wisselen. Daarna combineren we alles binnen de herhalingslus "Zolang *A* niet 0 is". Bekijk volgende code. Deze code werkt niet; zie jij waarom niet?

In [9]:

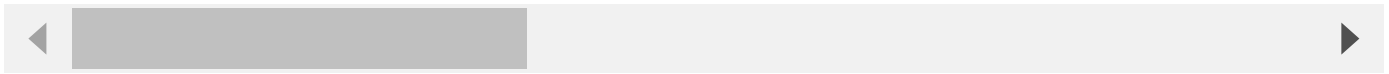
```
# initialisatie om te testen
A=5
B=10

# code om te wisselen
A=B
B=A

print(A,B)
```

10 10

Probeer eerst zelf te achterhalen wat er misloopt en hoe dit kan opgelost worden alvorens de oplossing hierna te bekijken. Je kan eventueel de [code in Python Tutor](http://www.pythontutor.com/visualize.html#code=%23%20initialisatie%20om%20te%20testen%0AA%3D5%0Afrontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false) (<http://www.pythontutor.com/visualize.html#code=%23%20initialisatie%20om%20te%20testen%0AA%3D5%0Afrontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false>) stap voor stap doorlopen.



**Oplossing:** het probleem is dat het commando  $A=B$  de waarde van  $A$  definitief overschrijft zodat die niet meer gebruikt kan worden in het commando  $B=A$ . Dit kan je vermijden door vooralleer je  $A$  overschrijft, de waarde te bewaren in een andere variabele, bijvoorbeeld *Temp*:

In [10]:

```
# initialisatie om te testen
A=5
B=10

# code om te wisselen
Temp=A
A=B
B=Temp

print(A,B)
```

10 5

### Alles samenvoegen in de herhalingslus

Nu hebben we alle stappen klaar en moeten we enkel nog de buitenste herhalingslus schrijven. Merk op dat we binnen een herhalingslus eender welk programmaatje kunnen neerzetten, zolang de indentatie maar correct is. Dus, volgende code wordt:

Zolang  $B$  niet 0 is, herhaal: Voer de herhalingslus van stap 2 uit Wissel  $A$  en  $B$  om (zodat  $A$  terug de grootste is)  
Het antwoord is  $B$  # input # stap 1 while  $B \neq 0$ : # "!=" betekent: "niet gelijk aan" # stap 2 # wissel  $A$  en  $B$  om  
`print(A)`

Als we dan alle stukjes code aanvullen, krijgen we uiteindelijk:

In [11]:

```
# input
X=49
Y=35

# stap 1
if X<Y:
    A=Y
    B=X
else:
    A=X
    B=Y

while B!=0:
    # stap 2
    while B<=A:
        A = A - B

    # wissel A en B om
    T=A
    A=B
    B=T

print(A)
```

7

Is de code nog niet helemaal duidelijk? Loop dat stap voor stap door de [code met Python Tutor](http://www.pythontutor.com/visualize.html#code=%23%20input%0AX%3D49%0AY%3D35%0A%0A%23%20s%20B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%23%20wissel%20A%20en%20B%20of%20frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false) (<http://www.pythontutor.com/visualize.html#code=%23%20input%0AX%3D49%0AY%3D35%0A%0A%23%20s%20B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%23%20wissel%20A%20en%20B%20of%20frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false>)!

## Conclusie

We beëindigen deze notebook met een herhaling van de belangrijkste punten die je leerde:

- Je maakte kennis met volgende constructies in Python:
  - variabelen
  - stapsgewijze uitvoering van commando blokken
  - conditionele uitvoering
  - herhalingslussen
- Je begrijpt de code die we schreven voor het algoritme van Euclides om de grootste gemene deler van twee getallen te berekenen.

Tijdens de practica sessies zal je via oefeningen van toenemende complexiteit leren om een probleem te analyseren, een stappenplan op te stellen (= *algoritme*) en dit vervolgens te vertalen naar een *implementatie* in Python.

In de volgende notebooks gaan we dieper in op de verschillende syntactische constructies waar we in deze notebook kennis mee maakten.