Pythoncursus

Opdrachtenserie 1

Tanja, Koen en Marein

september 2017

Voor deze opdracht ga je Python als rekenmachine gebruiken. Hierbij ga je een ABC-formule implementeren en breuken versimpelen.

Opdracht 1 - ABC-formule

De ABC-formule kennen jullie natuurlijk allemaal; hiermee kan je eenvoudig de oplossingen van een kwadratische vergelijking vinden. We gaan er voor deze opdracht even vanuit dat de discriminant altijd positief is ($D > 0$), omdat we nog niet geleerd hebben hoe we gevallen kunnen onderscheiden.

Mocht je het inmiddels vergeten zijn, hier nog even de formules. Voor een vergelijking van de vorm $ax^2 + bx + c = 0$ geldt dat $D = b^2 - 4ac$ en $x = \dfrac{-b \pm \sqrt{D}}{2a}$.

Neem nu de vergelijking $3x^2 + 3x - 5 = 0$. We gaan een paar regels Pythoncode schrijven om de oplossingen te vinden. Misschien zie je nu de oplossing al, of kan je het veel sneller op papier oplossen. Als je echter meerdere vergelijkingen wil oplossen is het een stuk makkelijker om je programmaatje aan te passen dan je berekening helemaal opnieuw uit te voeren.

a) Sla de waarden van de vergelijking op in een paar Python-variabelen. Zorg ervoor dat je de variabelen goede namen geeft, zodat het overzichtelijk blijft. In dit geval zijn de namen letters (`a`, `b`, `c`, etc.), maar over het algemeen wil je variabelenamen meer beschrijvend maken (`price`, `time`, `image`, etc.).

b) Maak nu een stukje Pythoncode om de oplossing van de vergelijking uit te rekenen met de ABC-formule. Het is handig om dit in twee stappen te doen, zoals je in de formule gewend bent: reken eerst D uit, en daarna de beide oplossingen voor x_0 en x_1. Zorg ervoor dat je programma de oplossingen op het scherm toont.

Tip: Voor het uitrekenen van de wortel kun je een ingebouwde functie in Python gebruiken: `sqrt(a,b)`. Zet hiervoor aan het begin van je programma `from math import sqrt`. Zie ook de Sectie *"Toelichting: Imports"* aan het einde van dit document.

Nog een paar vergelijkingen

Eén vergelijking kon je natuurlijk ook wel op papier, maar wat nu als je de volgende vergelijkingen allemaal wilt oplossen?

$$\begin{aligned} x^2 + 7x - 18 &= 0 \\ x^2 + 6.28318x - 9.86959 &= 0 \\ 3x^2 - 12x &= 0 \\ 2x^2 - 37x + 1 &= 0 \end{aligned}$$

c) Pas je programma zo aan dat het de antwoorden van de volgende vergelijkingen na elkaar print. Denk eraan dat, wanneer je variabelenamen hergebruikt, je de oude waarde overschrijft. Dit kan je op allerlei manieren voorkomen of oplossen. Kies zelf een oplossing die je het meest geschikt lijkt.

d) Kijk ook eens wat er gebeurt wanneer je de onderstaande vergelijking probeert op te lossen. Kan je verklaren waarom dit gebeurt?

$$x^2 + 5x + 7 = 0$$

Opdracht 2 - Breuken versimpelen

Zelf versimpelen

In deze opdracht ga je breuken versimpelen. Bij een breuk $\frac{a}{b}$ heb je hiervoor de grootste gemene deler nodig. Dit is het grootste getal, waardoor a en b allebei te delen zijn. Probeer zelf te bedenken hoe dit bruikbaar is. In het Engels heet dit de *greatest common divisor*, afgekort de *gcd*. De *gcd* van bijvoorbeeld 24 en 32 is bijvoorbeeld 8 omdat dit het grootste getal is waardoor zowel 24 als 32 deelbaar is. Voor het uitrekenen van de gcd kun je een ingebouwde functie in Python gebruiken als `gcd(a,b)`. Zet hiervoor aan het begin van je programma: `from fractions import gcd`.

a) Sla de breuk $\frac{a}{b}$ op door twee Pythonvariabelen `a` en `b` te maken. Je mag zelf twee waarden voor `a` en `b` kiezen.

Je kunt meerdere teksten en/of getallen achter elkaar printen door deze allemaal in één print-aanroep te zetten, gescheiden door een komma. Als `a` bijvoorbeeld 42 is, zal `print('a =', a)` de tekst *a = 42* laten zien.

b) Print je breuk nu volgens bovenstaande printaanroep. Probeer je twee variabelen nu zo te schrijven, dat het programma correct blijft werken als je a en b een andere waarde geeft, zonder dat je daarvoor de teksten in de `print` hoeft te veranderen.

c) Schrijf nu een programma dat begint met twee getallen $a = 1081$ en $b = 483$. Het programma moet het volgende afdrukken: "$1081/483 = \ldots$" met op de puntjes de versimpelde breuk. Je kunt hier de `gcd(a,b)`-functie gebruiken zoals aan het begin van deze opdracht is beschreven.

Ingebouwd in Python

Je kunt breuken ook automatisch door een Python-functionaliteit laten versimpelen. Gebruik hiervoor `from fractions import Fraction`. Door nu simpelweg `Fraction(a,b)` uit te rekenen krijg je automatisch een versimpelde breuk. Door deze te printen zie je de versimpelde breuk.

Bonus opdracht: `gcd` implementeren

Als je tot hier bent gekomen en je hebt nog tijd over, dan kun je misschien deze bonus opdracht proberen. Hiervoor heb je al wel wat stof nodig uit de volgende les, namelijk `while`-loops. Vraag hierover dus uitleg bij de assistenten.

De opdracht is om de `gcd` die je eerder hebt gebruikt nu zelf te schrijven. Begin hierbij weer met twee variabelen `a` en `b` die de breuk weergeven. Print aan het einde weer de versimpelde breuk. Maar de berekening moet nu dus zonder gebruik te maken van de voorgegeven `gcd`. Als je geen idee hebt hoe je dit zou moeten berekenen, vraag het de assistenten.

Toelichting: Imports

Je hebt bij de vorige opdrachten gezien dat we een paar functies nodig hadden die niet standaard toegankelijk zijn. We moesten ze eerst importeren uit andere modules. Zo kwam de functie `sqrt` uit de math-module, en kwam `gcd` uit `fractions`.

Met `import` kan je functies of variabelen uit een andere module importeren in het huidige programma. Dit is hoe in principe alle modules in Python zijn opgebouwd. Dit zorgt ervoor dat de code makkelijk herbruikbaar is. In een later stadium zullen we bespreken hoe je zelf een module kunt maken die anderen kunnen importeren.

Je kunt op twee manieren functies importeren. Zoals je hierboven zag kan je de constructie `from` *`modulenaam`* `import` *`functienaam`* gebruiken. Dan kan je de functie meteen gebruiken. Soms heb je meerdere functies met dezelfde naam uit verschillende modules. In dat geval kan je de module in z'n geheel importeren met `import` *`modulenaam`*, en een functie gebruiken door eerst de modulenaam te noemen. De `gcd`-functie uit de `fractions`-module zou je dan importeren met `import fractions`. Vervolgens kun je de `gcd`-functie dan als volgt gebruiken: *fractions.gcd(12, 32)*.

Als je een bepaalde functie zoekt is het handig om de Python-documentatie te bekijken, maar als je al ongeveer weet waar je moet zoeken kan het ook sneller. Na de module te importeren met `import` *`modulenaam`*, kun je met de functie `dir(`*`modulenaam`*`)` bekijken wat je er zoal uit kan gebruiken. Voor de bekendste modules is zelfs een handleiding ingebouwd. Deze kun je aanroepen met de `help`-functie (bij de math module zou dat `help(math)` worden). Probeer bijvoorbeeld eens de math-module te importeren en met `dir` te printen wat er allemaal voor functies inzitten.