Pythoncursus

Opdrachtenserie 2

Tanja, Koen en Marein

september 2017

Opdracht 1 - Geld maakt gelukkig

In deze opdracht gaan we aan de slag met geld. We representeren geld hier met twee variabelen, namelijk hele euro's en losse centen. We gaan stapsgewijs twee bedragen aan de gebruiker vragen en vervolgens bij elkaar optellen. Input van de gebruiker kun je krijgen met de `input()`-functie.

a) Bedenk een leuk bedrag en representeer die in je Pythonprogramma met twee variabelen (euro's en centen). Print dit bedrag op een nette manier (bijvoorbeeld: "Dat kost €10.17") zoals je in de vorige opdracht geleerd hebt.

b) Vraag de gebruiker met een `print` en `input` om ook een bedrag in te voeren. Je kunt eerst om hele euro's vragen en daarna om centen. Let er op dat als de gebruiker een ongeldig bedrag invult (bijvoorbeeld €15,-19 of een string tekst) dat je nogmaals om een geldig bedrag moet vragen. Denk hier aan de types van de variabelen (je moet dus controleren of de invoer van de gebruiker omgezet kan worden naar een integer). Je kunt hiervoor de boolean functie `isnumeric` gebruiken die je op strings kunt aanroepen, bijvoorbeeld: `'10'.isnumeric()` of `euros.isnumeric()`. Je hebt voor deze opdracht een loop nodig met boven in de loop een conditie die controleert of het bedrag correct is.

c) Tel de twee bedragen van (a) en (b) nu bij elkaar op en print het resultaat naar de gebruiker. Let er op dat als je over de "centenlimiet" heengaat, je een euro bij het resultaat op moet tellen. €4,80 + €2,25 is bijvoorbeeld €7,05. Je zult hier aan de slag moeten met if-statements.

Opdracht 2 - De rij van fibonacci

De rij van Fibonacci is een beroemde rij getallen waarmee heel wat problemen uit 'de echte wereld' gemodelleerd kunnen worden. Deze rij begint met 0 en 1. Ieder volgend element is de som van de twee voorgaande getallen. Het derde getal is dus de som van het eerste en tweede getal, dus $0 + 1 = 1$. Het vierde getal is dan $1 + 1 = 2$ en het vijfde getal dus $1 + 2 = 3$.

De rij van Fibonacci wordt ook wel de konijnenrij genoemd, omdat de manier waarop konijnen zich voortplanten hiermee (niet helemaal realistisch) beschreven kan worden. Het fibonaccigetal beschrijft het aantal konijnenparen, die zich voortplanten.

Een nieuw konijnenpaar is de eerste maand nog jong, maar kan in de tweede maand zelf ook nieuwe konijnenparen krijgen. We beginnen met 1 paar konijnen en de konijnen sterven nooit. Het aantal konijnen is dan steeds gelijk aan het aantal konijnen in de vorige maand, plus het aantal nieuwe konijnen: dit is het aantal konijnen twee maanden geleden. Elk paar dat er twee maanden geleden al was, is nu immers volwassen en krijgt een nieuw paar. Het aantal konijnen neemt hiermee exponentieel toe.

a) Maak een Pythonprogramma dat de eerste 20 getallen uit de Fibonacci-rij op het scherm tovert. Als je het slim doet heb je naast de loop-variabele maar twee andere variabelen nodig om de twee huidige getallen bij te houden. Hierbij is de constructie $a, b = 0, 1$ handig.

b) De ratio tussen twee opeenvolgende getallen uit de Fibonacci-rij (te schrijven als $\frac{f(i+1)}{f(i)}$) benadert een getal (de ratio 'convergeert': de ratio neigt dus naar een bepaald getal naarmate i groter wordt). Vind dit getal programmatisch. Weet je welke beroemde ratio dit is?

c) Experimenteer met verschillende beginwaarden voor a en b en kijk wat er gebeurt met de ratio.

Opdracht 3 - FizzBuzz

De afgelopen jaren zijn er steeds meer bedrijven bijgekomen die op zoek zijn naar programmeurs. De banen liggen voor het oprapen. Helaas zorgt dit ervoor dat steeds meer mensen claimen dat ze kunnen programmeren bij een sollicitatie, terwijl ze nog nooit een regel code hebben uitgevoerd. Zelfs wanneer mensen een prima gesprek kunnen voeren over programmeren, blijkt soms dat ze de grootste problemen hebben met een ogenschijnlijk simpel programmaatje. Er verschijnen steeds vaker allerlei blogposts waarin mensen het probleem samenvatten[1]: programmers just can't program.

Omdat jullie inmiddels al aardig wat ervaring hebben opgedaan, zou het volgende probleem gesneden koek moeten zijn. Je herkent het wellicht als kinderspelletje.

a) Schrijf een programma dat alle getallen van 1 tot 100 print. Echter, bij elke veelvoud van 3 ($3, 6, 9, 12..$) moet het programma '`Fizz`' printen in plaats van het getal, en bij elke veelvoud van 5 moet je '`Buzz`' printen. Een veelvoud van 3 en 5 print je als '`FizzBuzz`'. Probeer het programma zoveel mogelijk in één keer op te schrijven voordat je het test.

Opdracht 4 - Een wortel uitrekenen

Zoals je weet heeft Python een ingebouwde functie voor het uitrekenen van een wortel. Het is echter een goede oefening om zelf een programma te maken dat de wortel zelf kan benaderen. Dit kan op veel verschillende manieren. Het algoritme wat

[1] `http://blog.codinghorror.com/why-cant-programmers-program/`

wij gebruiken, werkt met een bovengrens en een ondergrens. De wortel ligt tussen deze twee waarden in. In een loop worden deze waarden steeds wat naar elkaar toegeschoven, zodat je steeds preciezer weet wat de wortel ongeveer is. **Zie ook de presentatie-slides voor een grafische uitleg**. Het algoritme begint als volgt:

- De gebruiker voert een getal in, die in deze uitleg x wordt genoemd. Hiervan ga je de wortel zoeken.
- Je mag een bepaalde foutmarge aanhouden - je programma mag er bijvoorbeeld 0,01 naast zitten. Je mag hiervoor een constante waarde in je programma zetten, of je mag het door de gebruiker laten intypen.
- Als je algoritme begint, kun je nog weinig zeggen over deze onder- en bovengrens. Je moet hiervoor dus veilige waarden nemen, waarvan je zeker weet dat de wortel ertussenin ligt. Voor de ondergens kun je nul nemen, en voor de bovengrens kun je x nemen. Dit werkt alleen niet als x tussen 0 en 1 ligt. Welk getal moet je dan gebruiken?
- Je weet dat $\sqrt{x}$ ergens tussen de onder- en bovengrens ligt, dus als schatting voor $\sqrt{x}$ kunnen we het gemiddelde van deze twee nemen.

a) Maak een programma, waarin deze beginwaarden goed worden gezet (x, foutmarge, onder- en bovengrens, schatting). Het algoritme zelf hoeft nog niet uitgevoerd te worden.

Nu heb je alle gegevens die je nodig hebt, en kan het zoeken beginnen. Dit gebeurt in een loopje:

- In de conditie van de while-loop controleren we hoe ver we van de echte wortel af zitten. Dit kunnen we doen door onze laatste schatting te kwadrateren, en met x te vergelijken.
- Als het kwadraat van onze laatste schatting groter is dan x zitten we kennelijk te hoog. Dit is nu een veilige nieuwe bovengrens. Als de schatting lager is, is het de nieuwe ondergrens.

Als uit de conditie van de while-loop blijkt dat onze schatting goed genoeg is, zijn we klaar.

b) Vul nu je programma aan, zodat dit algoritme wordt uitgevoerd. Laat de gebruiker een getal intypen, en eventueel een foutmarge als je hiervoor geen vaste waarde in je programma wil nemen. Voer het algoritme uit, en print de resulterende wortelbenadering.

   Hint Als je tijdens het programmeren een foutje maakt, kan het gebeuren dat de loop nooit eindigt. Druk in PyCharm op de stop-knop om je programma te stoppen. Door in de loop de waarden van je variabelen te printen, kun je zien hoe de variabelen veranderen. Dit kan helpen met ontdekken wat er fout gaat.

Opdracht 5 - Priemfactorontbinding

We gaan in deze opdracht een geheel getal ontbinden in zo klein mogelijke stukjes, die het oorspronkelijke getal geven als ze vermenigvuldigd worden. Zo is het getal 12 bijvoorbeeld te ontbinden in de getallen 2, 2 en 3, want $2 \cdot 2 \cdot 3 = 12$. De ontbinding 3 en 4 is fout, omdat 2 kleiner is en je deze dus liever in je ontbinding wil hebben dan 4. Het getal 1 gebruiken we niet in een ontbinding, dus 2 is het kleinste getal.

a) Schrijf een programma dat de gebruiker om een (positief) getal vraagt en de ontbinding van dit getal geeft. De uitvoer moet van de volgende vorm zijn, als de gebruiker bijvoorbeeld het getal 12 invoert:

```
12 = 2 * 2 * 3
```

Als je in Python twee prints wilt doen op dezelfde regel (en dus niet de tweede print op een nieuwe regel zoals normaal gesproken gebeurt), kun je de constructie `print(x, end="")` gebruiken, waarbij je op x je variabelen/strings die je wilt printen invoert. Als je een extra nieuwe regel wilt printen, moet je de string "\n " printen. *Let op!* Uit een gewone deling `a/b` komt een floating-point getal. Om het algoritme goed te laten werken heb je misschien een integer-deling nodig, `a//b`. Deze zorgt altijd voor exacte uitkomsten als het gaat om integers.

Misschien weet je al dat de getallen die hieruit komen priemgetallen zijn, oftewel getallen die alleen deelbaar zijn door 1 en zichzelf. Dit wordt dan ook de priemfactorontbinding genoemd (let op: je hoeft voor deze opdracht niet expliciet een lijst van priemgetallen te berekenen). In de cryptografie is de priemfactorontbinding van grote getallen een belangrijk probleem. Dit is niet erg efficient uit te rekenen.

b) Met `time.time()` kun je de huidige tijd opvragen (in seconden, sinds 1 januari 1970). Let erop dat je `time` importeert met `import`. Pas met behulp van deze functie je programma aan zodat aan het eind wordt geprint hoe lang de berekening geduurd heeft. Probeer daarna het getal 242971737361422649675227403 76 te ontbinden en meet hoe lang dat dit duurt. De computer heeft wat tijd nodig om dit getal te ontbinden!

In de cryptografie worden vaak getallen van meer dan 600 cijfers gebruikt. Om dit te ontbinden is een computer honderden jaren bezig.