Pythoncursus

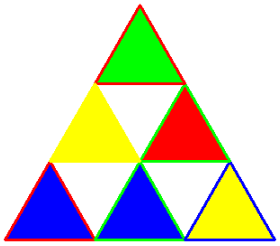
Opdrachtenserie 3

Tanja, Koen en Marein

september 2017

Opdracht 1 - Turtlegraphics

In deze opdracht gaan we tekenen! Hiervoor gebruiken we de library `turtlegraphics`, waarmee je met Python een schildpad kunt 'besturen', die vervolgens lijntjes tekent. We zullen stapsgewijs kleine tekeningetjes maken, waarbij we uiteindelijk de volgende figuur gaan tekenen:



Zoals je kunt zien bestaat de figuur uit heel veel driehoekjes. We willen de code om één driehoekje te tekenen maar één keer opschrijven. Daarom is het handig om hier een functie voor te maken. Als oefening gaan we eerst wat simpele dingen tekenen. Om te beginnen met tekenen moet je de volgende code bovenin je Python-programma zetten:

```
import turtle
#
# Zet hier je turtle-code
#
turtle.done()
```

De `done()` aan het einde van het programma zorgt ervoor dat het venster niet meteen afgesloten wordt. Om het venstertje van `turtlegraphics` af te sluiten kun je op de rode 'stopknop' van PyCharm drukken.

Nu heb je de beschikking over de schildpad! In tabel 1 kun je een aantal functies vinden die je kunt gebruiken om de schildpad te laten lopen en tekenen[1].

a) Teken twee losse vierkantjes met de schildpad, waarbij je gebruik maakt van de functies uit de tabel. Je mag de grootte van de vierkantjes zelf bepalen.

[1] Op `https://docs.python.org/3/library/turtle.html` kun je nog veel meer informatie en functies vinden!

`position()`	Vraag de positie op van de schilpad: je krijgt een x en een y terug.
`done()`	Pauzeert het programma aan het einde van het tekenen, zodat het venster niet meteen afgesloten wordt.
`forward(x)`	Laat de schildpad x stapjes zetten.
`left(x)`	Draai de schildpad x graden naar links. `right(x)` bestaat natuurlijk ook.
`reset(x)`	Maak de tekening leeg en zet de turtle weer in het midden.
`circle(x)`	Teken een cirkel met straal x
`setposition(x,y)`	Zet de schildpad op positie (x, y).
`speed(x)`	Zet de snelheid waarmee de schildpad loopt op x.

Tabel 1: Mogelijke functies voor de schildpad

Naast het 'saaie' zwart kun je ook andere kleuren gebruiken om te tekenen. Met `color()` kun je de huidige gebruikte kleuren opvragen. Uit deze functie komen twee waarden, waarbij de eerste de kleur van de pen is en de tweede de kleur waarmee je vlakken in kunt kleuren.

Je kunt de penkleur veranderen met `pencolor(x)` waarbij x een string is van de kleur (in het Engels). De invulkleur verander je met `fillcolor(x)`.

Je kunt gesloten vlakken (zoals een vierkant, cirkel of driehoek) ook automatisch inkleuren. Voordat je begint met tekenen van een vlak roep je de functie `begin_fill()` aan. Vervolgens teken je het vlak. Als je het hele vlak getekend hebt (en het vlak dus gesloten is aan alle kanten), roep je de functie `end_fill()` aan.

b) Vraag aan de gebruiker om twee kleuren en een afmeting in te vullen. Gebruik deze twee kleuren en de afmeting om een vierkant te tekenen. Het is de bedoeling dat je de kleuren in het Nederlands vraagt. Vervolgens moet je ze omzetten naar de Engelse benaming omdat `pencolor(x)` Engelse kleuren vereist. Zorg dat je programma in ieder geval *geel, groen, rood, blauw* en *paars* ondersteunt.

Nu we wat geoefend hebben met de schildpad gaan we de figuur tekenen. Deze figuur bestaat uit allemaal driehoekjes.

c) Probeer als oefening eerst een driehoekje te tekenen. Gebruik voor zowel de pen- als de fillcolor de kleur 'red'.

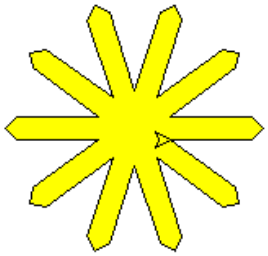
d) We moeten heel wat driehoekjes tekenen, maar we willen deze code maar eenmalig opschrijven. Maak daarom nu een functie `tekendriehoek(kleur)` die als parameter een kleur meekrijgt en vervolgens een driehoek met die kleur tekent.

Nu we met een enkele functie een driehoek kunnen tekenen is het tijd om de hele figuur te tekenen. Deze figuur bestaat uit zes driehoeken. De andere driehoeken bestaan uit verschillende kleuren. Let op dat sommige randen een andere kleur hebben dan de binnenkant! Je zult dus zowel de *pencolor* als de *fillcolor* aan moeten passen.

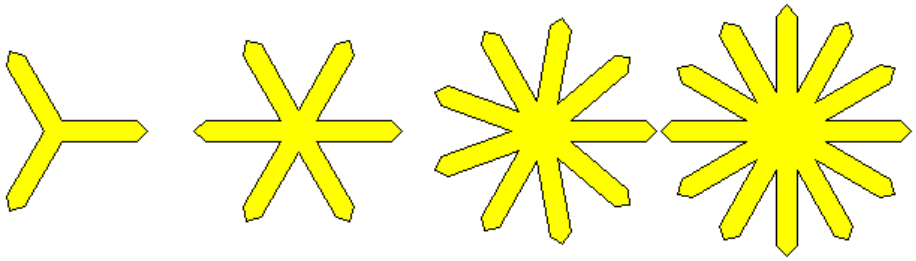
e) Teken nu alle driehoeken van de figuur op de juiste plaats, waarbij je de `tekendriehoek`-functie gebruikt die je bij (d) hebt gemaakt. Je zult hier een parameter aan moeten toevoegen voor de pencolor. Om de schildpad op de juiste plaats te zetten kun je

gebruik maken van `setposition(x,y)`.

f) Maak nu een functie, die het onderstaande 'zonnetje' tekent. Maak de functie zo dat het aantal uitsteeksels kan variëren. Hiervoor moet je nadenken over de hoek die je moet draaien, deze is afhankelijk van het aantal uitsteeksels. Misschien is het nodig dit eerst op papier uit te tekenen.



g) Gebruik de functie uit de vorige opdracht om de onderstaande afbeelding te maken. Werkt je oplossing ook goed, als je niet 4 maar 100 figuren zou moeten tekenen?



Opdracht 2 - Fair and Square

Google heeft regelmatig programmeerwedstrijden, waaraan iedereen mee mag doen[2]. Om te oefenen met functies maken we hiervan een opdracht. Deze opdracht gaat over getallen die palindromen zijn - dit zijn getallen die hetzelfde zijn, als je ze achterstevoren leest. Zo zijn 55 en 10801 palindromen, maar 123 en 40 niet (ook al kunnen we dit als 040 schrijven, dit tellen we niet mee).

a) Schrijf een functie die een palindroom kan herkennen. Deze functie moet een integer als invoer krijgen, en `True` of `False` opleveren.

b) Schrijf ook een functie die op dezelfde manier kan herkennen of een integer een kwadraat is.

c) We gaan nu uitzoeken hoeveel getallen binnen een bepaald bereik, zowel palindroom als kwadraat zijn. Maak een functie die twee getallen als invoer krijgt die het bereik aangeven, en aangeeft hoeveel getallen er voldoen.

[2] https://code.google.com/codejam/contests.html

d) Zoek nu uit hoeveel getallen er voldoen in de bereiken $(0, 100)$, $(100, 1000)$, en $(1000, 100000)$

Opmerking: bij de opdrachten van Google moet je je algoritme vaak loslaten op een gemakkelijke invoer, en op een moeilijke invoer. De gemakkelijke invoer kan op de bovenstaande manier, maar voor de moeilijke invoer moet je een slimmer en efficiënter algoritme bedenken!

Opdracht 3 - Galgje

In deze opdracht gaan we het galgje-spel bouwen. We zullen het stapje voor stapje opbouwen, maar je kunt zoiets natuurlijk zo uitgebreid maken als je zelf wilt. Als je merkt dat de opdracht toch nog wat lastig is kan je ervoor kiezen om de laatste delen voor nu achter wegen te laten. Probeer het steeds zo te programmeren dat je tussendoor zo veel mogelijk stukjes kunt testen zonder dat het spel helemaal af hoeft te zijn.

Voordat we gaan beginnen met het eigenlijke galgje-spel gaan we eerst even een hulpfunctie maken die je straks nodig gaat hebben. Zoals je in de huiswerkopdrachten gezien hebt kun je strings (stukjes tekst) niet eenvoudig bewerken. Toch is het voor galgje handig als we in een string een puntje zouden kunnen vervangen door een letter.

a) Maak nu de functie `vervang(woord, n, c)` die de letter op de *n*-de plek van het woord vervangt door de letter in de variabele `c`. Dit kan je bijvoorbeeld opdelen in een paar stappen: knip het woord in de stukken die je nodig hebt, plak ze weer aan elkaar en lever het resultaat als uitvoer op. Let op: we beginnen altijd bij 0 te tellen. `vervang('abcd', 1, 'e')` levert dus `aecd` op.

Als je ervan overtuigd bent dat je functie goed werkt gaan we aan het spel zelf beginnen. We beginnen met een paar variabelen waar we de toestand van het spel in opslaan.

b) Maak een paar variabelen aan met duidelijke namen om bij te houden wat het te raden woord is, hoeveel kansen de gebruiker nog heeft en wat het huidige geraden woord is. Kies logische beginwaarden; in eerste instantie een vast woord (bijvoorbeeld "radboud"). Voor ongeraden letters is het handig om puntjes te gebruiken.

Gedurende het spel zal de variabele die het huidige geraden woord bevat steeds meer gaan lijken op het te raden woord. We gaan natuurlijk de functie die je bij (a) hebt gemaakt gebruiken om steeds een letter te vervangen.

Het spel zelf komt eigenlijk neer op één grote loop. Bij elke iteratie van de loop is de gebruiker één keer aan de beurt.

c) Heb je hier een `while`-loop of een `for`-loop nodig? Aan welke condities moet zijn voldaan voordat de gebruiker nog een keer aan de beurt mag komen? Hoe bepaal je of 'ie niet al verloren heeft? En hoe bepaal je of 'ie gewonnen heeft? Combineer dit, en schrijf in één regel de 'kop' van de loop op – de inhoud komt zometeen.

Nu gaan we stapje voor stapje de loop invullen.

d) Toon de huidige speltoestand (aantal pogingen, geraden woord tot nu toe) aan de gebruiker, en vraag om een nieuwe letter. Je hoeft geen rekening te houden met foutieve input (getallen, hele zinnen), maar dat mag je natuurlijk wel proberen.

Aan de hand van de letter die de gebruiker heeft ingevuld kunnen we twee gevallen onderscheiden: de letter zit in het woord, of de letter zit niet in het woord. In Python kan je in een `if`-statement met de operator `in` controleren of een letter in een string voorkomt (en algemener: of een element in een lijst of verzameling voorkomt). Zo zal `if 'a' in 'abc'` als resultaat `True` opleveren, terwijl `if 'z' in 'abc'` juist `False` oplevert. Je kunt dit overigens ook gebruiken voor deelwoorden: `'bc' in 'abc'` is waar, maar `'ac' in 'abc'` of `'ba' in 'abc'` niet.

e) Onderscheid deze twee gevallen, en vul alvast in wat er moet gebeuren wanneer de gebruiker een letter raadt die niet in het woord voorkomt.

Wanneer een letter wel in het woord voorkomt moeten we uitzoeken waar de letter precies staat, en de letter op die plek(ken) invullen in het 'tot nu toe geraden woord'. Voor dat laatste hebben we gelukkig al een functie gemaakt..

f) Ga nu één voor één na welke letters overeen komen met de letter die de gebruiker heeft ingevuld. Bedenk je wat de functie die je bij (a) hebt gemaakt precies nodig heeft, en pas 'm toe om de letter op de juiste plaats(en) in te vullen in het geraden woord.

Hint: In plaats van over een `range()` te lopen in een loop kun je ook het statement `enumerate()` gebruiken. Voer de volgende code eens uit en kijk wat er gebeurt:

```
for i,j in enumerate('hoi'):
    print(i,j)
```

Waarschijnlijk kun je dit soort constructies ook nuttig gebruiken in je code.

Dit is het einde van de loop. Als je je conditie bij (c) goed hebt gekozen zou het spel nu door moeten gaan als de gebruiker nog een beurt heeft, en moeten stoppen wanneer hij wint of verliest.

g) Zorg voor een net bericht aan het einde wanneer de gebruiker heeft gewonnen of verloren. Waar kan je aan zien of de gebruiker won of verloor? Onderscheid deze twee gevallen. Laat ook nog even zien wat het te raden woord was.

Het zou goed kunnen dat je er al rekening mee hebt gehouden, maar de kans is aanwezig dat je code nu alleen werkt voor woorden van een bepaalde vaste lengte.

h) Indien je dit nog niet gedaan hebt, pas je code zo aan dat het werkt voor een woord van elke willekeurige lengte. Gebruik hiervoor de functie `len(woord)` in plaats van een constante waarde.

bonus 1 Wanneer de gebruiker een bepaalde letter al geraden heeft, is het een beetje zonde als het nog een gok kost wanneer hij of zij die letter per ongeluk herhaalt. Hoe kan je handig bijhouden welke letters al geraden zijn? Bedenk wanneer je een letter toe moet voegen, en voorkom dat een letter twee keer fout gerekend wordt.

bonus 2 Allemaal leuk en aardig, maar je wist het woord aan het begin natuurlijk al omdat je het zelf in hebt gevuld. In plaats daarvan kan je ook een willekeurig

woord uit een file lezen. Omdat dat nog niet behandeld is, geven we die code hieronder voor. Kan je verklaren wat er hier gebeurt? Op de site staat een woordenlijst die je zou kunnen gebruiken, maar je kan natuurlijk ook zelf wat woorden in een tekstbestand zetten. Zorg ervoor dat er steeds maar één woord per regel staat.

```
import random
bestand = open('woordenlijst.txt')
woorden = bestand.read().splitlines()
woord = random.choice(woorden)
```

bonus 3 Je kunt ook de gebruiker vragen om een woord in te vullen. Zo kan je het spel met twee mensen spelen. Zorg er wel voor dat je het scherm even leeg maakt, zodat de ander niet ziet wat je hebt ingevuld. Dat kan natuurlijk met een aantal prints, maar dat kan ook netter. Met de juiste zoektermen in Google moet je het juiste commando wel kunnen vinden.

Opdracht 4 - Teken de galg

Het doel van deze opdracht is het tekenen van de galg die je zou verwachten bij het spel uit opdracht 2. Hiervoor gaan we de Turtlegraphics uit opdracht 1 weer gebruiken.

a) Combineer wat je geleerd hebt bij opdracht 1 met het spel wat je gemaakt hebt bij opdracht 2. Ga na waar je precies een stukje moet tekenen. Om je code overzichtelijk te houden is het goed om hier een aparte functie van te maken. Welke informatie heeft deze functie allemaal nodig? Hoe kan je deze functie op een nette manier opschrijven, zodat de galg-tekening leesbaar blijft? Introduceer een schaalfactor zodat je je galg makkelijk van grote kan veranderen. Misschien kan je zelfs variabelen maken voor lengtes die je meerdere keren aflegt, zodat je bij een wijziging maar een getal hoeft te veranderen.

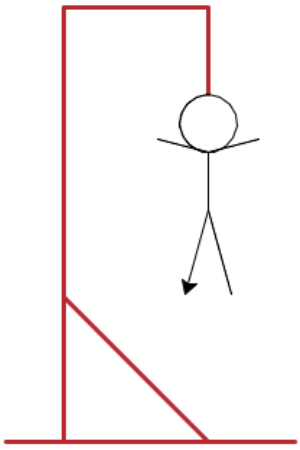
In het algemeen is het een goede gewoonte om de code die je gebruikt voor het weergeven van informatie (zoals het tekenen van de galg) zoveel mogelijk te scheiden van de logica. Dit maakt het makkelijker om later één van de twee aan te passen zonder dat dat de ander sloopt.

b) Python heeft een bijzonder keyword dat je in weinig andere talen tegenkomt: `yield`. Hiermee kan je een functie onderbreken (en eventueel een resultaat opleveren). In tegenstelling tot `return` kan de functie in een later stadium gewoon weer verder waar 'ie gebleven was. Hoe het technisch precies werkt is op dit moment niet heel belangrijk, maar het kan interessant zijn om het een keer gezien te hebben.

   De eerste keer dat je de functie aanroept wordt er een 'generator' gemaakt. Deze moet je opslaan in een variabele. Om vervolgens de functie uit te voeren tot de volgende `yield` kan je `next()` uitvoeren op de generator – je geeft de generator dus als argument aan `next()`. Aannemende dat `f` een functie is die `yield`s bevat ziet het er bijvoorbeeld als volgt uit. De functie tekenGalg() wordt nu drie keer hervat tot de volgende `yield`. In dit voorbeeld heet de generator `galg`.

```
galg = tekenGalg()
next(galg)
next(galg)
next(galg)
```

Kan je dit patroon toepassen op je functie voor het tekenen van de galg? Als je dit goed doet hoeft je functie niet eens meer te weten hoeveel pogingen de gebruiker nog heeft; er is helemaal geen argument meer nodig.



Figuur 1: Een voorbeeld van een leuke galg

Bonusopdracht - Teken een dorpje

Dit is een bonusopdracht voor als je nog veel tijd over hebt en het leuk vindt om te tekenen.

Maak het onderstaande dorpje. Er zijn wat elementen die vaker voorkomen: hier kun je functies voor maken, die je dan meerdere keren kan aanroepen.