In dit document vind je de oplossingen van de programmeeroefeningen aan het einde van de hoofdstukken uit het boek. Er is niet altijd maar één juist antwoord van toepassing op een programmeeroefening. Het kan dus best zo zijn dat jouw antwoord niet overeenkomt met de oplossing in dit document. De voorbeelden geven je een idee van mogelijke oplossingen. Op de website van het boek vind je ook de code bestanden van deze oplossingen die je kunt downloaden.

HOOFDSTUK 3

#1: FAVORIETEN

Hier zie je een oplossing met drie favoriete spelletjes en drie favoriete gerechten:

```
>>> spelletjes = ['Pokemon', 'Lego Mindstorms', 'Pacman']
>>> eten = ['Pannenkoeken', 'Chocola', 'Appels']
>>> favorieten = spelletjes + eten
>>> print(favorieten)
['Pokemon', 'Lego Mindstorms', 'Pacman', 'Pannenkoeken',
'Chocola', 'Appels']
```

#2: STRIJDERS TELLEN

We kunnen deze berekening op een aantal manieren doen. We hebben 3 gebouwen met 25 ninja's die zich op elk dak hebben verstopt en 2 tunnels met in elke tunnel 40 samoerai. We kunnen het totale aantal ninja's en het totale aantal samoerai uitrekenen en dan deze twee getallen optellen:

```
>>> 3*25
75
>>> 2*40
80
>>> 75+80
155
```

Veel korter (en beter) is het om deze drie vergelijkingen te combineren door gebruik te maken van haakjes. Deze haakjes zijn niet per se noodzakelijk vanwege de volgorde van de berekeningen, maar ze maken het lezen van de vergelijking makkelijker:

```
>>> (3*25)+(2*40)
```

Misschien is onderstaande een beter antwoord. Deze code vertelt ons wat we berekenen:

```
>>> daken = 3
>>> ninjas_per_dak = 25
>>> tunnels = 2
>>> samoerai_per_tunnel = 40
>>> print((daken * ninjas_per_dak) + (tunnels *
samoerai_per_tunnel))
155
```

#3: GROETJES!

In dit antwoord geven we de variabelen die naar jouw voor- en achternaam verwijzen. Vervolgens zie je de string waarin plaatshouders staan om jouw naam te printen in een bericht waarbij die twee variabelen zijn gebruikt:

```
>>> voornaam = 'Brammetje'
>>> achternaam = 'Jansen'
>>> print('Hee hallo, %s %s!' % (voornaam, achternaam))
Hee hallo, Brammetje Jansen!
```

HOOFDSTUK 4

#1: EEN RECHTHOEK

Een rechthoek teken je bijna hetzelfde als een vierkant. Alleen bij een rechthoek moet de turtle twee zijden tekenen die groter zijn dan de andere twee zijden:

```
>>> import turtle
>>> t = turtle.Pen()
>>> t.forward(100)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(100)
>>> t.left(90)
>>> t.forward(50)
```

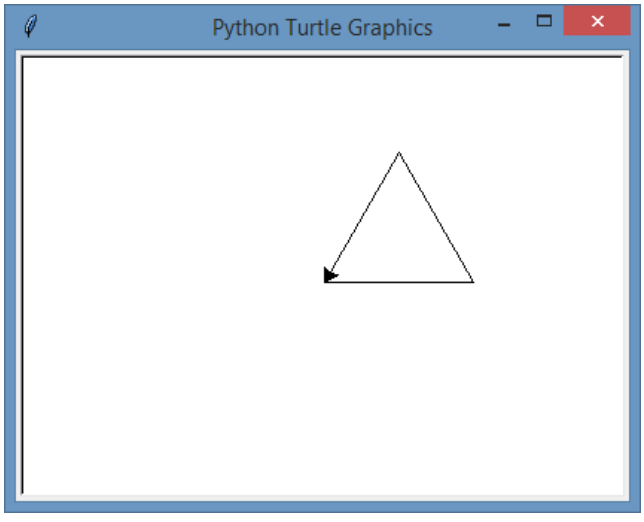
#2: EEN DRIEHOEK

De programmeeroefening gaf niet aan wat voor soort driehoek er getekend moest worden. Er zijn drie soorten driehoeken: *gelijkzijdig*, *gelijkbenig* en *ongelijkzijdig*. Afhankelijk van jouw kennis over geometrie, heb je een driehoek getekend. Mogelijk ging dit niet direct goed, maar was je uiteindelijk tevreden met hetgeen je gemaakt had. Voor dit voorbeeld concentreren we ons op het tekenen van de eerste twee typen driehoeken omdat deze het eenvoudigst zijn om te tekenen. Een gelijkzijdige driehoek heeft drie gelijke zijden en drie gelijke hoeken:

```
>>> import turtle
>>> t = turtle.Pen()
❶ >>> t.forward(100)
❷ >>> t.left(120)
❸ >>> t.forward(100)
```

```
❹ >>> t.left(120)
❺ >>> t.forward(100)
```

We tekenen de basis van de driehoek door 100 pixels vooruit te gaan bij ❶. We draaien 120 graden naar links (hiermee maak je een hoek van 60 graden) bij ❷, en vervolgens gaan we weer 100 pixels vooruit bij ❸. De volgende draai is ook 120 graden bij ❹, en de turtle gaat terug naar het start punt door nogmaals 100 pixels naar voren te gaan bij ❺. Hier is het resultaat van het uitvoeren van de code:



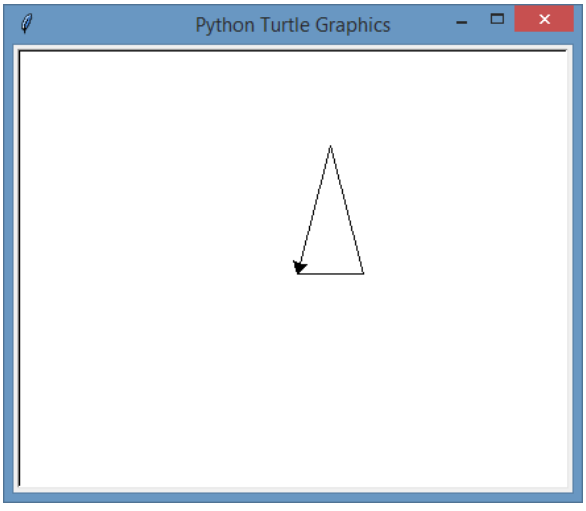
Een gelijkbenige driehoek heeft twee gelijke zijden en twee gelijke hoeken:

```
>>> import turtle
>>> t = turtle.Pen()
>>> t.forward(50)
>>> t.left(104.47751218592992)
>>> t.forward(100)
>>> t.left(151.04497562814015)
>>> t.forward(100)
```

In deze oplossing gaat de turtle 50 pixels vooruit en draait dan 104.47751218592992 graden. Het gaat 100 pixels vooruit, gevolgd door een draaiing van 151.04497562714015 graden, en gaat dan weer 100 pixels vooruit. Om de turtle in de start positie te zetten, roepen we de volgende regel opnieuw aan:

```
>>> t.left(104.47751218592992)
```

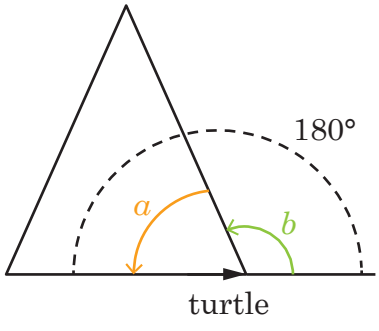
Hier is het resultaat van de uitgevoerde code:



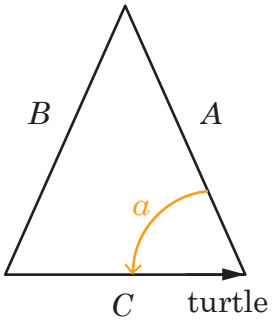
Hoe komen we aan hoeken met 104.47751218592992 en 151.04497562814015 graden? Immers, dit zijn best rare nummers.

Nadat we hebben besloten hoe lang elke zijde van de driehoek moet worden, kunnen we de hoeken van de driehoek berekenen door gebruik te maken van Python en een beetje meetkunde.

In de volgende afbeelding zie je dat als we de afmeting van hoek a weten, we na kunnen gaan wat de graden zijn van de buitenste driehoek b wat de turtle moet draaien. De twee hoeken a en b lopen op tot 180 graden.



Het is niet moeilijk om de binnenste driehoek uit te rekenen als je de juiste formule weet. We willen bijvoorbeeld een driehoek maken met een onderzijde van 50 pixels (laten we dit C noemen), en de twee zijdes A en B zijn beide 100 pixels lang.



De formule om de binnenste driehoek a te berekenen door middel van het gebruik van de zijden A, B, en C is:

$$a = \arccos\left(\frac{A^2 + C^2 - B^2}{2AC}\right)$$

We kunnen een klein Python-programma maken door gebruik te maken van de waarde van Pythons `math` module:

```
>>> import math
>>> A = 100
>>> B = 100
>>> C = 50
❶ >>> a = math.acos((math.pow(A, 2) + math.pow(C, 2) - \
        math.pow(B, 2)) / (2*A*C))
>>> print(a)
1.31811607165
```

Eerst importeren we de `math` module, en daarna maken we variabelen voor elke zijden (`A`, `B`, en `C`). Bij ❶ gebruiken we de `math` functie `acos` (arc cosine) om de hoek te berekenen. Deze berekening geeft de radiaal waarde 1.31811607165. Radialen zijn een andere eenheid om hoeken te meten, zoals graden ook gebruikt wordt.

LET OP!

De schuine streep (\) (backslash) in de code bij ❶ hoort niet bij de formule. Schuine strepen, zoals beschreven in Hoofdstuk 16 in het boek, worden gebruikt om lange regels met code te splitsen. Dit is niet verplicht, maar voor de leesbaarheid heb ik de schuine streep ingevoegd omdat de code anders niet op de pagina past.

De radiale waarde kan worden omgezet naar graden door gebruik te maken van de math functie `degrees`. Dan kunnen we de buitenste driehoek berekenen (de te berekenen waarde die we nodig hebben voor het draaien van de turtle), door dit getal van de 180 graden af te halen:

```
>>> print(180 - math.degrees(a))
104.477512186
```

De formule voor de volgende draai van de turtle is vergelijkbaar:

$$b = \arccos\left(\frac{A^2 + B^2 - C^2}{2AB}\right)$$

De code voor deze vergelijking lijkt ook gelijkwaardig:

```
>>> b = math.acos((math.pow(A,2) + math.pow(B,2) - \
        math.pow(C,2)) / (2*A*B))
>>> print(180 - math.degrees(b))
151.04497562814015
```

Natuurlijk hoef je niet per se deze codes te gebruiken om de hoeken uit te rekenen. Je kunt ook proberen en spelen met het aantal graden dat de turtle moet draaien, net zo lang tot je iets hebt dat er goed uitziet.

#3: EEN DOOS ZONDER HOEKEN

De oplossing van deze programmeeroefening (een achthoek met vier zijden die ontbreken) is eigenlijk vier keer hetzelfde. Ga vooruit, draai 45 graden naar links, til de pen op, ga vooruit, zet de pen neer en draai opnieuw 45 graden naar links:

```
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
```

Dus de laatste combinatie van instructies is de volgende code (de beste manier om te draaien is om een nieuw venster in de shell te maken en deze op te slaan als *geenhoeken.py*):

```
import turtle
t = turtle.Pen()
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
```

HOOFDSTUK 5

#1: BEN JE RIJK?

Je krijgt een *inspringingsfout* zodra je de laatste regels van het `if` statement hebt bereikt:

```
>>> geld = 2000
>>> if geld > 1000:
❶         print("Ik ben rijk!!")
else:
          print("Ik ben niet rijk!!")
❷         print("Maar misschien komt dat nog...")
SyntaxError: unexpected indent
```

Je krijgt deze foutmelding omdat het eerste blok bij ❶ begint met vier spaties, dus Python verwacht geen twee extra spaties op de laatste regel bij ❷. Het markeert de plek waar het een probleem ziet met een verticaal rechthoekig blok. Zo zie je direct waar er iets mis gaat.

#2: MARS

De code om na te gaan of het aantal Mars-repen minder dan 100 is, of groter dan 500 zou er zo uit moeten zien:

```
>>> marsen = 600
>>> if marsen < 100 or marsen > 500:
>>>     print('Te weinig of te veel')

Te weinig of te veel
```

#3: PRECIES HET GOEDE GETAL

Je kunt meer dan één `if` statement maken om te controleren of de hoeveelheid geld tussen 100 en 500 of 1000 en 5000 ligt, maar door gebruik te maken van de sleutelwoorden `and` en `or`, kan het wel met één statement:

```
if (geld >= 100 and geld <= 500) or (geld >= 1000 \
      and geld <= 5000):
    print('geld is tussen 100 & 500 of tussen 1000 & 5000')
```

(Zorg ervoor dat je haakjes plaatst rondom de eerste en laatste twee voorwaarden zodat Python nagaat of de hoeveelheid geld tussen de 100 en 500 *of* 1000 en 5000 ligt.)

We kunnen de code testen door de variabele `geld` op verschillende waarden te zetten:

```
>>> geld = 800
>>> if (geld >= 100 and geld <= 500) or (geld >= 1000 \
            and geld <= 5000):
        print('geld is tussen 100 & 500 of tussen 1000 & 5000')

>>> geld = 400
>>> if (geld >= 100 and geld <= 500) or (geld >= 1000 \
            and geld <= 5000):
        print('geld is tussen 100 & 500 of tussen 1000 & 5000')

geld is tussen 100 & 500 of tussen 1000 & 5000
>>> geld = 3000
>>> if (geld >= 100 and geld <= 500) or (geld >= 1000 \
            and geld <= 5000):
```

```
        print('geld is tussen 100 & 500 of tussen 1000 & 5000')

geld is tussen 100 & 500 of tussen 1000 & 5000
```

#4: DE NINJA'S KAN IK WEL AAN

Dit was een beetje een gemene programmeeroefening. Als je het `if` statement in dezelfde volgorde als de instructies maakt, krijg je waarschijnlijk niet het resultaat dat je verwachtte. Hier is een voorbeeld:

```
>>> ninjas = 5
>>> if ninjas < 50:
        print("Dat zijn er teveel")
elif ninjas < 30:
        print("Het zal lastig worden, maar ik kan ze wel aan")
elif ninjas < 10:
        print("Ik kan ze allemaal aan!")

Dat zijn er teveel
```

Zelfs als het aantal ninja's minder is dan 10, krijg je de boodschap "Dat zijn er teveel." Dit komt door de eerste voorwaarde (`< 50`) die als eerste door Python is beoordeeld. Omdat deze variabele echt minder dan 50 is, print het programma een boodschap die je niet verwacht te zien.

Om te zorgen dat het goed werkt, draai je de volgorde om waarin je het getal controleert, zodat je eerst ziet of het getal kleiner is dan 10:

```
>>> ninjas = 5
>>> if ninjas < 10:
        print("Ik kan ze allemaal aan!")
elif ninjas < 30:
        print("Het zal lastig worden, maar ik kan ze wel aan")
elif ninjas < 50:
        print("Dat zijn er teveel")

Ik kan ze allemaal aan!
```

HOOFDSTUK 6

#1: DE HALLO LUS

Het `print` statement in deze `for` lus wordt slechts één keer uitgevoerd. Dit komt doordat Python direct uit de lus springt zodra het programma bij `if` statement is aangekomen, omdat x dan kleiner is dan 9.

```
>>> for x in range(0, 20):
        print('hallo %s' % x)
if x < 9:
    break

hallo 0
```

#2: EVEN GETALLEN

We kunnen de step parameter met de `range` functie gebruiken om een lijst met even getallen te maken. Als je 14 jaar oud bent, is de start parameter 2, en de eind parameter is 16 (omdat de `for` lus doorgaat tot de waarde net voor de eind parameter).

```
>>> for x in range(2, 16, 2):
        print(x)
2
4
6
8
10
12
14
```

#3: VIJF VAN MIJN LIEVELINGSINGREDIËNTEN

Er is een aantal verschillende manieren om een lijst met nummers met verschillende ingrediënten te printen. Dit is één manier:

```
>>> ingredienten = ['slakken', 'bloedzuigers', 'navelschilfers van
                    gorilla', 'rupswenkbrauwen', 'duizendpootnagels']
❶ >>> x = 1
❷ >>> for i in ingredienten:
❸         print('%s %s' % (x, i))
❹         x = x + 1

1 slakken
2 bloedzuigers
3 navelschilfers van gorilla
4 rupswenkbrauwen
5 duizendpootnagels
```

We maken een variabele `x` om het getal op te slaan dat we willen printen bij ❶. Daarna maken we een `for` lus om de onderdelen in de lijst te doorlopen bij ❷, waarbij elke aan de variabele `i` wordt toegewezen en dan printen we de waarde van de `x` en `i` variabelen bij ❸, met gebruik van de `%s` plaatshouder. Bij ❹ vermeerderen we de `x` variabele met 1, zodat het aantal dat we printen bij elke herhaling van de lus groter wordt.

#4: WAT WEEG JIJ OP DE MAAN

Om je gewicht in kilo's over 15 jaar op de maan te berekenen, maak je eerst een variabele om je startgewicht op te slaan:

```
>>> gewicht = 30
```

Voor elk jaar kun je het nieuwe gewicht berekenen door een kilo toe te voegen, en vervolgens te vermenigvuldigen met 16,5% (0.165):

```
>>> gewicht = 30
>>> for jaar in range(1, 16):
        gewicht = gewicht + 1
        maangewicht = gewicht * 0.165
        print('Jaar %s is %s' % (jaar, maangewicht))

Jaar 1 is 5.115
Jaar 2 is 5.28
Jaar 3 is 5.445
Jaar 4 is 5.61
Jaar 5 is 5.775
Jaar 6 is 5.94
Jaar 7 is 6.105
Jaar 8 is 6.2700000000000005
Jaar 9 is 6.4350000000000005
Jaar 10 is 6.6000000000000005
Jaar 11 is 6.765000000000001
Jaar 12 is 6.930000000000001
Jaar 13 is 7.095000000000001
Jaar 14 is 7.260000000000001
Jaar 15 is 7.425000000000001
```

HOOFDSTUK 7

#1: BASISFUNCTIE VOOR HET GEWICHT OP DE MAAN

De nieuwe functie moet twee parameters nemen: `gewicht` en `toename` (de hoeveelheid gewicht zal elk jaar toenemen). De rest van de code is gelijk aan de oplossing van programmeeroefening #4 uit Hoofdstuk 6.

```
>>> def bereken_gewicht(gewicht, toename):
        for jaar in range(1, 16):
            gewicht = gewicht + toename
            maangewicht = gewicht * 0.165
            print('Jaar %s is %s' % (jaar, maangewicht))
>>> bereken_gewicht(40, 0.5)
Jaar 1 is 6.6825
Jaar 2 is 6.765
Jaar 3 is 6.8475
Jaar 4 is 6.93
Jaar 5 is 7.0125
Jaar 6 is 7.095
Jaar 7 is 7.1775
Jaar 8 is 7.26
Jaar 9 is 7.3425
Jaar 10 is 7.425
Jaar 11 is 7.5075
Jaar 12 is 7.59
Jaar 13 is 7.6725
Jaar 14 is 7.755
Jaar 15 is 7.8375
```

#2: MAANGEWICHT FUNCTIE EN JAREN

We hebben alleen een kleine verandering in de functie nodig zodat het aantal jaren kan worden doorgegeven als een parameter.

```
>>> def bereken_gewicht(gewicht, toename, jaren):
        jaren = jaren + 1
        for jaar in range(1, jaren):
            gewicht = gewicht + toename
            maangewicht = gewicht * 0.165
            print('Jaar %s is %s' % (jaar, maangewicht))

>>> bereken_gewicht(35, 0.3, 5)
Jaar 1 is 5.8245
```

```
Jaar 2 is 5.874
Jaar 3 is 5.9235
Jaar 4 is 5.973
Jaar 5 is 6.0225
```

Merk op dat we in de tweede regel van de functie 1 hebben toegevoegd aan de `jaren` parameter, zodat de `for` lus eindigt bij het juiste jaar (in plaats van bij het vorige jaar).

#3: MAANGEWICHT PROGRAMMA

We kunnen het `stdin` object van de `sys` module gebruiken om iemand toestemming te geven om waarden in te voeren (door middel van de `readline` functie). Omdat `sys.stdin.readline` een string geeft, is het nodig om deze strings te converteren naar nummers zodat we de berekeningen kunnen laten uitvoeren.

```
import sys
def bereken_gewicht():
    print('Voer het startgewicht in')
❶   gewicht = float(sys.stdin.readline())
    print('Voer de toename per jaar in')
❷   toename = float(sys.stdin.readline())
    print('Voer het aantal jaren in')
❸   jaren = int(sys.stdin.readline())
    jaren = jaren + 1
    for jaar in range(1, jaren):
        gewicht = gewicht + toename
        maangewicht = gewicht * 0.165
        print('Jaar %s is %s' % (jaar, maangewicht))
```

Bij ❶ lezen we de input door gebruik te maken van `sys.stdin.readline` en de string vervolgens te converteren naar een float, door de `float` functie te gebruiken. Deze waarde wordt opgeslagen als `gewicht` variabele. We doen hetzelfde bij ❷ voor de variabele `toename`, maar gebruiken de `int` functie bij ❸, omdat we alleen gehele getallen voor jaartallen invoeren (geen breuken). De rest van de code na die regel is precies hetzelfde als in de vorige oplossing.

Als we de functie uitvoeren, zien we zoiets als dit:

```
>>> bereken_gewicht()
Voer het startgewicht in
45
Voer de toename per jaar in
0.4
Voer het aantal jaren in
12
Jaar 1 is 7.491
Jaar 2 is 7.557
Jaar 3 is 7.623
Jaar 4 is 7.689
Jaar 5 is 7.755
Jaar 6 is 7.821
Jaar 7 is 7.887
Jaar 8 is 7.953
Jaar 9 is 8.019
Jaar 10 is 8.085
Jaar 11 is 8.151
Jaar 12 is 8.217
```

HOOFDSTUK 8

#1: DE GIRAFFE SHUFFLE

Voor het toevoegen van de functies om Ruger te laten bewegen, bekijken we de klassen `Dieren`, `Zoogdieren` en `Giraffen`.

Hier is de `Dieren` klasse (dit was een kind klasse van de `Levend` klasse, welke is verwijderd om dit voorbeeld iets makkelijker te maken):

```
class Dieren:
    def ademen(self):
        print('ademend')
    def bewegen(self):
        print('bewegend')
    def voedsel_eten (self):
        print('voedsel etend')
```

De klasse `Zoogdieren` is een kind klasse van `Dieren`:

```
class Zoogdieren(Dieren):
    def jongen_voeden_met_melk(self):
        print('jongen voedend')
```

En de klasse `Giraffen` is een kind klasse van `Zoogdieren`:

```
class Giraffen(Zoogdieren):
    def bladeren_van_bomen_eten(self):
        print('bladeren etend')
```

De functies voor het bewegen van elke voet zijn makkelijk toe te voegen:

```
class Giraffen(Zoogdieren):
    def bladeren_van_bomen_etend(self):
        print('bladeren etend')
    def linkervoet_voorwaarts(self):
        print('linkervoet naar voren')
    def rechtervoet_voorwaarts(self):
        print('rechtervoet naar voren')
    def linkervoet_achterwaarts(self):
        print('linkervoet naar achter')
    def rechtervoet_achterwaarts(self):
        print('rechtervoet naar achter')
```

De `dans` functie moet nu gewoon elke functie van de voeten in de juiste volgorde aanroepen:

```
    def dans(self):
        self.linkervoet_voorwaarts()
        self.linkervoet_achterwaarts()
        self.rechtervoet_voorwaarts()
        self.rechtervoet_achterwaarts()
        self.linkervoet_achterwaarts()
        self.rechtervoet_achterwaarts()
        self.rechtervoet_voorwaarts()
        self.linkervoet_voorwaarts()
```

Om Rutger te laten dansen, maken we een object en roepen we de functie aan:

```
>>> rutger = Giraffen()
>>> rutger.dans()

linkervoet naar voren
linkervoet naar achter
rechtervoet naar voren
rechtervoet naar achter
linkervoet naar achter
rechtervoet naar achter
rechtervoet naar voren
linkervoet naar voren
```

#2: TURTLE HOOIVORK

`Pen` is een gedefinieerde klasse in de `turtle` module, dus we kunnen meer dan één object maken met de `Pen` klasse voor alle vier de turtles. Als we ieder object toewijzen aan een andere variabele, kunnen we deze apart besturen. Dat maakt het makkelijk om de lijnen in deze oefening opnieuw te tekenen.

Het gegeven dat elk *object* een op zichzelf staand ding is, is erg belangrijk, met name wanneer we weer praten over klassen en objecten.

```
import turtle
t1 = turtle.Pen()
t2 = turtle.Pen()
t3 = turtle.Pen()
t4 = turtle.Pen()
t1.forward(100)
t1.left(90)
t1.forward(50)
t1.right(90)
t1.forward(50)
t2.forward(110)
t2.left(90)
t2.forward(25)
t2.right(90)
t2.forward(25)
t3.forward(110)
t3.right(90)
t3.forward(25)
t3.left(90)
t3.forward(25)
t4.forward(100)
t4.right(90)
t4.forward(50)
t4.left(90)
t4.forward(50)
```

Er zijn verschillende manieren om hetzelfde te tekenen, dus misschien ziet jouw code er anders uit dan bovenstaande code.

HOOFDSTUK 9

#1: MYSTERIEUZE CODE

De `abs` functie geeft de absolute waarde van een nummer, wat eigenlijk betekent dat een negatief nummer een positief nummer wordt. Dus in deze mysterieuze code geeft het eerste `print` statement 20 en de tweede geeft 0.

```
❶ >>> a = abs(10) + abs(-10)
  >>> print(a)
  20

❷ >>> b = abs(-10) + -10
  >>> print(b)
  0
```

De berekening bij ❶ eindigt als 10 + 10. De berekening bij ❷ verandert in 10 + –10.

#2: EEN VERBORGEN BOODSCHAP

De truc is om eerst een string te maken die de boodschap bevat. Daarna gebruik je de `dir` functie om na te gaan welke functies beschikbaar zijn in de string:

```
>>> boodschap = 'dit als is jij niet zijn een lezen heel dit
goede dan manier jij om hebben een doen boodschap ook te fout
verbergen'
>>> print(dir(boodschap))
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__',
'__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
'__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '_formatter_field_name_split',
'_formatter_parser', 'capitalize', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'index',
'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
```

Als je naar de lijst kijkt, lijkt de `split` functie handig. We kunnen de `help` functie gebruiken om na te gaan wat het doet:

```
>>> help(boodschap.split)
Help on built-in function split:

split(...)
    S.split([sep[, maxsplit]]) -> list of strings

    Return a list of the words in S, using sep as the
    delimiter string. If maxsplit is given, at most
    maxsplit splits are done. If sep is not specified
    or is None, any whitespace string is a separator
    and empty strings are removed from the result.
```

Volgens deze beschrijving geeft `split` de string terug als allemaal aparte, gescheiden woorden, aan de hand van de tekens die in de `sep` parameter worden benoemd. Als er geen `sep` parameter is, gebruikt de functie blanco spaties als scheidingsteken. Dus deelt deze functie onze string op in stukjes.

Nu we weten welke functie we moeten gebruiken, kunnen we de woorden in de string doorlopen in een lus. Er zijn een paar verschillende manieren om steeds een woord over te slaan bij het printen (dus om en om een woord printen, te beginnen bij het eerste woord). Hier is één van de mogelijkheden:

```
❶ >>> boodschap = 'dit als is jij niet zijn een lezen heel dit
goede dan manier jij om hebben een doen boodschap ook te fout
verbergen'
❷ >>> woorden = boodschap.split()
❸ >>> for x in range(0, len(woorden), 2):
❹       print(woorden[x])
```

We maken de string bij ❶ en bij ❷ gebruiken we de `split` functie om de string te verdelen in een lijst met losse woorden. Daarna maken we een `for` lus door de `range` functie te gebruiken bij ❸. De eerste parameter bij deze functie is 0 (het begin van de lijst). De volgende parameter gebruikt de `len` functie om de lengte van de lijst te bepalen (dit is dan het einde van het bereik). En de laatste parameter is een step waarde van 2 (op deze manier ziet de lijst van getallen eruit als 0, 2, 4, 6, 8, enzovoorts). We gebruiken de x variabele van de `for` lus om bij ❹ waarden uit de lijst te printen.

#3: EEN BESTAND KOPIËREN

Om een bestand te kopiëren, moet je het bestand dat je wilt kopiëren eerst openen, het lezen en dan een nieuw bestand maken: de kopie.

We openen het doelbestand om hier naar toe te schrijven (met gebruik van de `'w'` parameter om te schrijven) en dan schrijven we de inhoud van de variabele naar dit bestand. De definitieve code ziet er zo uit:

```
f = open('c:\\Users\\<jouw gebruikersnaam>\\Documents\\test.txt')
s = f.read()
f.close()
f = open('output.txt', 'w')
f.write(s)
f.close()
```

Dit voorbeeld werkt, maar het is beter om een bestand te kopiëren door gebruik te maken van de Python module met de naam `shutil`:

```
import shutil
shutil.copy('c:\\Users\\<jouw gebruikersnaam>\\Documents\\test.txt',
'output.txt')
```

HOOFDSTUK 10

#1: GEKOPIEERDE AUTO'S

Er zitten twee `print` statements in deze code en we moeten uitvinden wat er bij elk statement geprint wordt. Dit is de eerste:

```
>>> auto1 = Auto()
❶ >>> auto1.wielen = 4
❷>>> auto2 = auto1
>>> auto2.wielen = 3
>>> print(auto1.wielen)

3
```

Waarom is het resultaat van het `print` statement 3, als we duidelijk bij `auto1` 4 aangeven ❶? Dat komt omdat bij ❷ de beide variabelen `auto1` en `auto2` verwijzen naar hetzelfde object.

Nu gaan we kijken naar het tweede `print` statement:

```
>>> auto3 = copy.copy(auto1)
>>> auto3.wielen = 6
>>> print(auto1.wielen)

3
```

In dit voorbeeld is `auto3` een kopie van het object. Het duidt niet hetzelfde object aan als `auto1` en `auto2`. Dus wanneer we het aantal wielen op 6 zetten, heeft het geen effect op het aantal wielen van `auto1`.

#2: PICKLE FAVORIETEN

We gebruiken de `pickle` module om de inhoud van een variabele (of variabelen) in een bestand op te slaan:

```
❶>>> import pickle
❷>>> favorieten = ['PlayStation', 'Drop', 'Films', 'Python for Kids']
❸>>> f = open('favorieten.dat', 'wb')
❹>>> pickle.dump(favorieten, f)
 >>> f.close()
```

We importeren de `pickle` module bij ❶, en maken onze lijst met favoriete dingen bij ❷. Daarna `openen` we een bestand, genaamd *favorieten.dat* door de string `'wb'` als tweede parameter door te geven bij ❸ (dit betekent 'write-binary', schrijf binair). Vervolgens gebruiken we de `dump` functie van de `pickle` module om de inhoud van de variabele `favorieten` op te slaan bij ❹.

Het tweede deel van deze oplossing is om het bestand weer in te lezen. Ervan uitgaande dat je de shell hebt gesloten en weer geopend, moeten we de `pickle` module opnieuw importeren.

```
>>> import pickle
>>> f = open('favorieten.dat', 'rb')
>>> favorieten = pickle.load(f)
>>> print(favorieten)

['PlayStation', 'Drop', 'Films', 'Python for Kids']
```

Dit komt overeen met de andere code, behalve dat we het bestand nu openen met de parameter `'rb'` (wat binair lezen betekent), en we de `load` functie in de `pickle` module gebruiken.

HOOFDSTUK 11

#1: TEKEN EEN ACHTHOEK

Een achthoek heeft acht zijden. Dus we hebben op zijn minst een `for` lus nodig om een achthoek te tekenen. Als je even nadenkt over de richting van de turtle en wat hij moet doen tijdens het tekenen van de achthoek, kom je er misschien achter dat de pijl van de turtle rond zal draaien, zoals een klok, tegen de tijd dat hij klaar is met tekenen. Dit betekent dat het volledig 360 graden is gedraaid. Als we 360 delen door het nummer van de zijden van de achthoek, krijgen we het aantal graden van de hoek dat de turtle moet draaien na elke stap van de lus (45 graden, zoals in de hint staat).

```
>>> import turtle
>>> t = turtle.Pen()
>>> def achthoek(grootte):
        for x in range (1,9):
            t.forward(grootte)
            t.right(45)
```

We kunnen de functie aanroepen om deze te testen met 100 als grootte van een van de zijden:

```
>>> achthoek(100)
```

#2: TEKEN EEN DICHTE ACHTHOEK

Als we de functie aanpassen zodat het een gevulde achthoek tekent, maken we het moeilijker door er een lijntje omheen te tekenen. Een betere aanpak is om hier een parameter in te voegen om na te gaan of de achthoek moet worden gevuld.

```
>>> import turtle
>>> t = turtle.Pen()
>>> def achthoek(grootte, gevuld):
❶       if gevuld == True:
❷           t.begin_fill()
❸       for x in range(1,9):
            t.forward(grootte)
            t.right(45)
❹       if gevuld == True:
❺           t.end_fill()
```

Eerst controleren we of de `filled` parameter op `True` staat bij ❶. Als dat zo is, laten we de turtle weten dat hij het vullen kan starten door gebruik te maken van de `begin_fill` functie bij ❷. Daarna tekenen we de achthoek op de volgende twee regels. Op dezelfde manier als programmeeroefening 1. Daarna controleren we of de `filled` parameter op `True` staat bij ❸. Als dat zo is, roepen we de `end_fill` functie aan bij ❹, welke eigenlijk de vorm vult.

We kunnen deze functie testen door de kleur geel in te stellen en de functie aan te roepen waarbij de parameter op `True` staat (dus het zal vullen). We kunnen daarna de kleur weer op zwart zetten en de functie opnieuw aanroepen met de parameter op `False` voor de lijn er omheen.

```
>>> t.color(1, 0.85, 0)
>>> achthoek(40, True)
>>> t.color(0, 0, 0)
>>> achthoek(40, False)
```

#3: NOG EEN FUNCTIE OM STERREN MEE TE TEKENEN

De truc van deze sterren functie is om 360 graden te verdelen in een aantal punten, welke de binnenste hoek voor elke punt van de ster aangeeft (zie regel ❶ in de volgende code). Om de buitenste hoek te bepalen, trekken we dat aantal van 180 af om het nummer te krijgen van de hoeveelheid graden die de turtle moet draaien bij ❷.

```
import turtle
t = turtle.Pen()
def teken_ster(grootte, punten):
❶     hoek = 360 / punten
❷     for x in range(0, punten):
❸         t.forward(grootte)
❹         t.left(180 - hoek)
❺         t.forward(grootte)
❻         t.right(180-(hoek * 2))
```

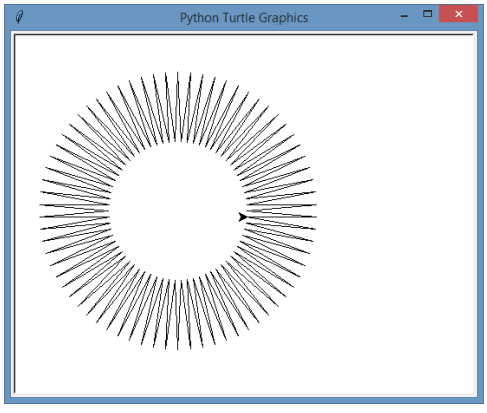
We doorlopen de lus van 0 tot het getal bij ❷ en vervolgens verplaatsen we de turtle het aantal pixels vooruit dat is aangegeven in de `size` parameter bij ❸. We draaien de turtle het aantal graden dat we hiervoor hebben berekend bij ❹ en gaan vervolgens nogmaals vooruit bij ❺ die de eerste “stekel” van de ster tekent. Om te

bewegen in een cirkelvormig patroon tijdens het tekenen van de stekels, moeten we de hoek verhogen, zodat we de uitgerekende hoek vermenigvuldigen met twee en de turtle naar rechts laten draaien bij ❻.

Je kunt deze functie bijvoorbeeld aanroepen met 80 pixels en 70 punten:

```
>>> teken_ster(80, 70)
```

Dan krijg je dit resultaat:



HOOFDSTUK 12

#1: VUL HET SCHERM MET DRIEHOEKEN

Om het scherm met driehoeken te vullen, is de eerste stap om een doek te tekenen. In dit voorbeeld geven we een hoogte en breedte van het doek van 400 pixels.

```
>>> from tkinter import *
>>> import random
>>> w = 400
>>> h = 400
>>> tk = Tk()
>>> canvas = Canvas(tk, width=w, height=h)
>>> canvas.pack()
```

Een driehoek heeft drie punten, wat betekent drie groepen van x- en y-coördinaten. We kunnen de `randrange` functie in de `random` module gebruiken (zoals in the willekeurige driehoek in Hoofdstuk 12), om willekeurig de coördinaten van de drie punten te verkrijgen (zes nummers

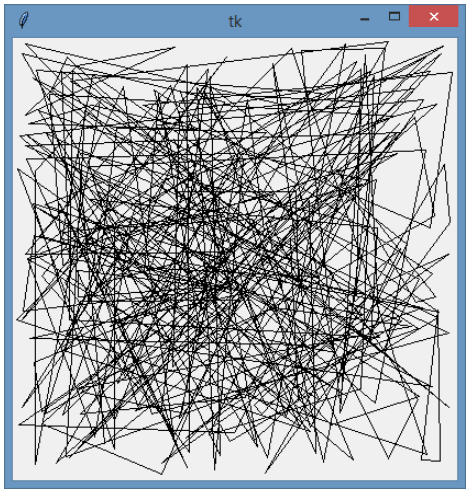
in totaal). Daarna gebruiken we de `willekeurige_driehoek` function om de driehoek te tekenen.

```
>>> def willekeurige_driehoek():
        p1 = random.randrange(w)
        p2 = random.randrange(h)
        p3 = random.randrange(w)
        p4 = random.randrange(h)
        p5 = random.randrange(w)
        p6 = random.randrange(h)
        canvas.create_polygon(p1, p2, p3, p4, p5, p6, \
                fill="", outline="black")
```

Uiteindelijk maken we een lus om heel veel willekeurige driehoeken te tekenen.

```
>>> for x in range(0, 100):
        willekeurige_driehoek()
```

Je ziet zoiets als dit:



Om het doek met willekeurig gekleurde driehoeken te vullen, maak je eerst een lijst met kleuren. We voegen dit toe aan de voorbereidende code aan het begin van het programma.

```
>>> from tkinter import *
>>> import random
>>> w = 400
>>> h = 400
>>> tk = Tk()
>>> canvas = Canvas(tk, width=w, height=h)
>>> canvas.pack()
>>> colors = ['red','green','blue','yellow','orange','white','purple']
```

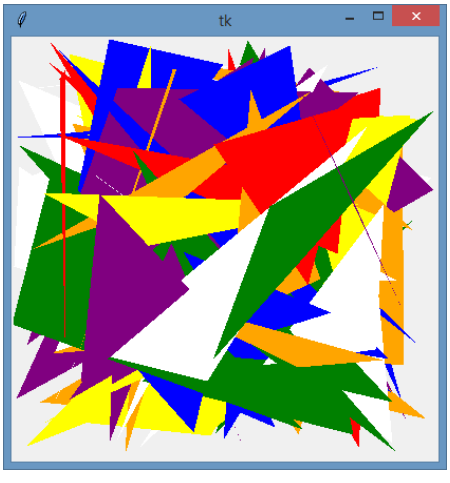
We kunnen de `choice` functie van de `random` module gebruiken om willekeurig een item van de lijst met kleuren te kiezen en het te gebruiken om `create_polygon` aan te roepen:

```
def willekeurige_driehoek():
    p1 = random.randrange(w)
    p2 = random.randrange(h)
    p3 = random.randrange(w)
    p4 = random.randrange(h)
    p5 = random.randrange(w)
    p6 = random.randrange(h)
    color = random.choice(colors)
    canvas.create_polygon(p1, p2, p3, p4, p5, p6, \
            fill=color, outline="")
```

Stel dat we de lus opnieuw 100 keer aanroepen:

```
>>> for x in range(0, 100):
        willekeurige_driehoek()
```

Het resultaat moet zoiets als dit zijn:



#2: DE BEWEGENDE DRIEHOEK

Voor de bewegende driehoek maken we weer eerst een doek. Daarna tekenen we de driehoek door gebruik te maken van de `create_polygon` functie:

```
import time
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=200, height=400)
canvas.pack()
canvas.create_polygon(10, 10, 10, 60, 50, 35)
```

Om de driehoek horizontaal over het doek te laten bewegen, moet de x-coördinaat een positief nummer zijn en het y-coördinaat moet 0 zijn. We maken hier een `for` lus voor, door gebruik te maken van 1 als ID van de driehoek, 10 voor de x parameter en 0 voor de y parameter:

```
for x in range(0, 35):
    canvas.move(1, 10, 0)
    tk.update()
    time.sleep(0.05)
```

Naar beneden over het scherm bewegen gaat bijna op dezelfde manier, met een 0 waarde voor de x parameter en een positieve waarde voor de y parameter:

```
for x in range(0, 14):
    canvas.move(1, 0, 10)
    tk.update()
    time.sleep(0.05)
```

Om weer terug over het scherm te bewegen, hebben we een negatieve waarde nodig voor de x parameter (en weer een 0 voor de y parameter). Om naar boven te bewegen hebben we een negatieve waarde nodig voor de y parameter.

```
for x in range(0, 35):
    canvas.move(1, -10, 0)
    tk.update()
    time.sleep(0.05)

for x in range(0, 14):
    canvas.move(1, 0, -10)
    tk.update()
    time.sleep(0.05)
```

#3: DE BEWEGENDE FOTO

De code voor de bewegende foto hangt af van het formaat van je afbeelding. We gaan ervan uit dat je afbeelding de naam *gezicht.gif* heeft en je het hebt opgeslagen op de harde schijf (C:)

Op de computer kun je de afbeelding weergeven en laten bewegen, net als elke andere getekende vorm.

```
import time
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
```

```
canvas.pack()
mijnfoto = PhotoImage (file='c:\\Users\\<gebruikersnaam>\\ \
Documents\\gezicht.gif')
canvas.create_image(0, 0, anchor=NW, image=mijnfoto)
for x in range(0, 35):
    canvas.move(1, 10, 10)
    tk.update()
    time.sleep(0.05)
```

Deze code laat de afbeelding diagonaal over het scherm gaan.

Als je een Mac OS X computer gebruikt, moet je een andere locatie invoeren dan in het Windows voorbeeld. Het ziet er dan als volgt uit:

```
myimage = PhotoImage(file='/Users/sarawinters/gezicht.gif')
```

HOOFDSTUK 14

#1: VERTRAAG DE START VAN HET SPEL

Om het spel te laten starten als de je op de doek klikt, moeten we een paar kleine aanpassingen in het spel maken. Eerst voegen we een nieuwe functie aan de `Paddle` klasse toe:

```
    def turn_left(self, evt):
        self.x = -2

    def turn_right(self, evt):
        self.x = 2
    def start_game(self, evt):
        self.started = True
```

Deze functie zal de object variabele `started` op `True` zetten wanneer hij wordt aangeroepen. We moeten deze object variabele ook in de `__init__` functie van `Paddle` inbouwen (en hem op `False` zetten) en een event binding toevoegen voor de functie `start_game` (waarmee we de start van het spel aan de muisknop koppelen).

```
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10,
        fill=color)
        self.canvas.move(self.id, 200, 300)
```

```
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
❶       self.started = False
        self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        self.canvas.bind_all('<KeyPress-Right>',
        self.turn_right)
❷       self.canvas.bind_all('<Button-1>', self.start_game)
```

Je ziet de toevoeging van de nieuwe object variabele `started` bij ❶ en de event binding voor de muisknop bij ❷.

De laatste wijziging wordt aangebracht in de laatste lus in de code. We moeten controleren of de object variabele `started` op `True` staat, voordat we de bal en paddle tekenen. Deze controle zie je in dit `if` statement.

```
while 1:
    if bal.hit_bottom == False and paddle.started == True:
        bal.draw()
        paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

#2: EEN ECHTE "GAME OVER"

We kunnen de `create_text` functie gebruiken om de tekst "Game Over" te tonen wanneer de bal de onderkant van het scherm geraakt heeft. We voegen dit toe net na de code voor het maken van een bal en paddle.

```
paddle = Paddle(canvas, 'blue')
bal = Bal(canvas, paddle, 'red')
spel_over_text = canvas.create_text(250, 200, text='GAME OVER', \
        state='hidden')
```

De `create_text` functie bevat een parameter met de naam `state`, welke we toevoegen aan de string `'hidden'`. Dit betekent dat Python de tekst tekent, maar het onzichtbaar maakt. Om de tekst te tonen als het spel afgelopen is, voegen we aan het einde van de code een nieuw `if` statement aan de lus toe.

```
while 1:
    if bal.hit_bottom == False and paddle.started == True:
        bal.draw()
        paddle.draw()
❶   if bal.hit_bottom == True:
❷       time.sleep(1)
```

```
❸          canvas.itemconfig(spel_over_text, state='normal')
        tk.update_idletasks()
        tk.update()
        time.sleep(0.01)
```

We zien of de object variabele `hit_bottom` op `True` staat bij ❶. Als dat zo is, wachten we voor 1 seconde bij ❷ (om een korte vertraging van het weergeven van de tekst te geven) en daarna veranderen we de `state` parameter van de tekst naar `'normal'` in plaats van `'hidden'` bij ❸, door de `itemconfig` functie van het doek te gebruiken.

We geven aan deze functie twee parameters door: het identificatienummer van de tekst die op het doek is getekend (opgeslagen in de variabele `game_over_text`) en de benoemde parameter `state`.

#3: VERSNEL DE BAL

Deze verandering is eenvoudig, maar je vond het mogelijk moeilijk om uit te vinden waar in de code je de verandering moet toevoegen. We willen dat de bal sneller beweegt als hij in dezelfde horizontale richting beweegt wanneer hij de paddle heeft geraakt en we willen dat hij langzamer gaat als hij in tegenovergestelde richting beweegt. Om dit te bereiken moeten we de links-rechts (horizontale) snelheid van de paddle optellen bij de horizontale snelheid van de bal.
De makkelijkste plaats om deze verandering toe te voegen is in de `hit_paddle` functie in de klasse `Bal`:

```
    def hit_paddle(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
❶           if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
❷               self.x += self.paddle.x
                return True
        return False
```

Wanneer we eenmaal hebben vastgesteld dat de bal de paddle heeft geraakt bij ❶, tellen we de waarde van de `x` variabele van het paddle object op bij de `x` variabele van de bal, bij ❷. Als de paddle over het scherm naar rechts beweegt (als bijvoorbeeld zijn `x` variabele `2` is) en de bal met een `x` waarde van `3` naar rechts beweegt en tegen de paddle aankomt, zal de bal van de paddle afkaatsen met een nieuwe (horizontale) snelheid van 5. Wanneer we beide `x` variabelen optellen betekent dit dat de bal een nieuwe

snelheid krijgt nadat hij de paddle heeft geraakt.

#4: HOUD DE SCORE VAN DE SPELER BIJ

Om de score toe te voegen aan het spel, maken we een nieuwe klasse met de naam `Score`:

```
class Score:
    def __init__(self, canvas, color):
❶       self.score = 0
❷       self.canvas = canvas
❸       self.id = canvas.create_text(450, 10, text=self.score, \
                                     fill=color)
```

De `__init__` functie van de klasse `Score` heeft drie parameters: `self`, `canvas`, en `color`. De eerste regel van deze functie maakt een object variabele `score`, met een waarde van 0 bij ❶. We maken ook de `canvas` parameter om later te gebruiken als de object variabele `canvas` bij ❷.

We gebruiken de `canvas` parameter om de score tekst te maken op positie (450, 10) en stellen de vulling van de waarde in van de `color` parameter bij ❸. De tekst is de huidige waarde van de variabele `score` (in andere woorden, 0).

De klasse `Score` heeft een andere functie nodig, welke gebruikt zal worden om de score te verhogen en de nieuwe waarde weer te geven.

```
class Score:
    def __init__(self, canvas, color):
        self.score = 0
        self.canvas = canvas
        self.id = canvas.create_text(450, 10, text=self.score,
            \ fill=color)

❶   def hit(self):
❷       self.score += 1
❸       self.canvas.itemconfig(self.id, text=self.score)
```

De `hit` functie heeft geen parameters ❶ en verhoogt de score met 1 bij ❷, voordat de `itemconfig` functie van het `canvas` object wordt gebruikt om de weergegeven tekst te veranderen in de nieuwe score bij ❸.

We kunnen een object van de klasse `Score` te maken net voordat we de `paddle` en `bal` objecten maken:

```
score = Score(canvas, 'green')
paddle = Paddle(canvas, 'blue')
bal = Bal(canvas, paddle, score, 'red')
spel_over_text = canvas.create_text(250, 200, text='GAME OVER', \
        state='hidden')
```

De laatste verandering in de code is in de klasse `Bal`. We moeten het object `Score` opslaan (welke we gebruiken wanneer we het `Bal` object maken) en daarna de `hit` functie activeren in de `hit_paddle` functie van de bal.

Het begin van de `Bal`s `__init__` functie heeft nu de parameter `score`, welke we gebruiken om een object variabele te maken. Deze heet ook `score`.

```
    def __init__(self, canvas, paddle, score, color):
        self.canvas = canvas
        self.paddle = paddle
        self.score = score
```

De `hit_paddle` functie zou er nu zo uit moeten zien:

```
    def hit_paddle(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
                self.x += self.paddle.x
                self.score.hit()
                return True
        return False
```

De volledige code voor dit spel moet er, nadat je alle vier de programmeeroefeningen hebt gedaan, zo uitzien:

```
from tkinter import *
import random
import time
```

```
class Bal:
    def __init__(self, canvas, paddle, score, color):
        self.canvas = canvas
        self.paddle = paddle
        self.score = score
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 245, 100)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()
        self.hit_bottom = False

    def hit_paddle(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
                self.x += self.paddle.x
                self.score.hit()
                return True
        return False

    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 3
        if pos[3] >= self.canvas_height:
            self.hit_bottom = True
        if self.hit_paddle(pos) == True:
            self.y = -3
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.canvas_width:
            self.x = -3

class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
        self.started = False
        self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        self.canvas.bind_all('<KeyPress-Right>', self.turn_right)
        self.canvas.bind_all('<Button-1>', self.start_spel)
```

```
        def draw(self):
            self.canvas.move(self.id, self.x, 0)
            pos = self.canvas.coords(self.id)
            if pos[0] <= 0:
                self.x = 0
            elif pos[2] >= self.canvas_width:
                self.x = 0

        def turn_left(self, evt):
            self.x = -2

        def turn_right(self, evt):
            self.x = 2

        def start_game(self, evt):
            self.started = True

    class Score:
        def __init__(self, canvas, color):
            self.score = 0
            self.canvas = canvas
            self.id = canvas.create_text(450, 10, text=self.score, \
              fill=color)

        def hit(self):
            self.score += 1
            self.canvas.itemconfig(self.id, text=self.score)

    tk = Tk()
    tk.title("Spel")
    tk.resizable(0, 0)
    tk.wm_attributes("-topmost", 1)
    canvas = Canvas(tk, width=500, height=400, bd=0, \
    highlightthickness=0)
    canvas.pack()
    tk.update()

    score = Score(canvas, 'green')
    paddle = Paddle(canvas, 'blue')
    bal = Bal(canvas, paddle, score, 'red')
    spel_over_text = canvas.create_text(250, 200, text='GAME OVER', \
    state='hidden')

    while 1:
        if bal.hit_bottom == False and paddle.started == True:
            bal.draw()
            paddle.draw()
        if bal.hit_bottom == True:
            time.sleep(1)
            canvas.itemconfig(spel_over_text, state='normal')
        tk.update_idletasks()
```

```
        tk.update()
        time.sleep(0.01)
```

HOOFDSTUK 16

#1: DAMBORD

Om een dambord als achtergrondafbeelding te tekenen, moeten we de lussen in de `__init__` functie van ons spel als volgt veranderen:

```
        self.bg = PhotoImage(file="achtergrond.gif")
        w = self.bg.width()
        h = self.bg.height()
❶       draw_background = 0
        for x in range(0, 5):
            for y in range(0, 5):
❷               if draw_background == 1:
❸                   self.canvas.create_image(x * w, y * h, \
                    image=self.bg, anchor='nw')
❹                   draw_background = 0
❺               else:
❻                   draw_background = 1
```

Bij ❶ maken we de variabele `draw_background` en zetten de waarde hiervan op 0. Bij ❷ controleren we of de waarde van de variabele 1 is, en als dat zo is, tekenen we de achtergrondafbeelding bij ❸ en zetten de variabele weer op 0 bij ❹. Als de waarde geen 1 is (dat is `else` bij ❺), zetten we de waarde op 1 bij ❻.
Wat doet deze verandering met de code? De eerste keer dat we het `if` statement aanroepen, tekent het geen achtergrondafbeelding en staat `draw_background` op 1. De volgende keer dat we het `if` statement aanroepen, tekenen we een afbeelding en zetten we de variabele terug op 0.
Elke keer dat we de lus doorlopen, keren we de waarde van de variabele om. De ene keer tekenen we de afbeelding, de volgende keer doen we dit niet.

#2: DAMBORD MET WISSELENDE AFBEELDING

Als je eenmaal ontdekt hebt hoe je het dambordeffect maakt, twee afwisselende beelden tekenen in plaats van een afbeelding en een lege, is het vrij gemakkelijk. We moeten de nieuwe achtergrondafbeelding toevoegen, net als

de originele afbeelding. In het volgende voorbeeld, voegen we de nieuwe afbeelding *achtergrond2.gif* toe. (Je moet deze wel eerst in GIMP maken) en slaan dit dan op als object variabele `bg2`.

```
        self.bg = PhotoImage(file="achtergrond.gif")
        self.bg2 = PhotoImage(file="achtergrond2.gif")
        w = self.bg.width()
        h = self.bg.height()
        draw_background = 0
        for x in range(0, 5):
            for y in range(0, 5):
                if draw_background == 1:
                    self.canvas.create_image(x * w, y * h, \
                            image=self.bg, anchor='nw')
                    draw_background = 0
                else:
                    self.canvas.create_image(x * w, y * h, \
                            image=self.bg2, anchor='nw')
                    draw_background = 1
```

In het tweede deel van het `if` statement dat we hebben gemaakt in de eerste programmeeroefening gebruiken we de `create_image` functie om de nieuwe afbeelding in het venster te tekenen.

#3: BOEKENPLANK EN LAMP

Om verschillende achtergronden te tekenen, kunnen we beginnen met onze afwisselende dambord code, maar we zullen het weer veranderen om een paar nieuwe afbeeldingen toe te voegen en deze dan in het doek te plaatsen. Voor dit voorbeeld kopieer je eerste de afbeelding *achtergrond2.gif* en teken er dan een eenvoudige boekenplank op. Deze nieuwe afbeelding sla je op als *plank.gif*. Daarna maak je nog een kopie van *achtergrond2.gif*, teken je een lamp en geeft deze afbeelding de naam *lamp.gif*.

```
        self.bg = PhotoImage(file="achtergrond.gif")
        self.bg2 = PhotoImage(file="achtergrond2.gif")
❶       self.bg_shelf = PhotoImage(file="plank.gif")
❷       self.bg_lamp = PhotoImage(file="lamp.gif")
        w = self.bg.width()
        h = self.bg.height()
❸       count = 0
            draw_background = 0
            for x in range(0, 5):
                for y in range(0, 5):
                    if draw_background == 1:
                        self.canvas.create_image(x * w, y * h, \
```

```
                                image=self.bg, anchor='nw')
                        draw_background = 0
                else:
❹                   count = count + 1
❺                   if count == 5:
❻                       self.canvas.create_image(x * w, y * h, \
                            image=self.bg_shelf, anchor='nw')
❼                   elif count == 9:
❽                       self.canvas.create_image(x*w, y*h, \
                                image=self.bg_lamp, anchor='nw')
                    else:
                        self.canvas.create_image(x * w, y * h, \
                                image=self.bg2, anchor='nw')
                    draw_background = 1
```

We voegen de nieuwe afbeeldingen toe bij ❶ en ❷ en slaan deze op als de variabelen `bg_shelf` en `bg_lamp`. Daarna maken we een nieuwe variabele `count` bij ❸. In de vorige oplossing hadden we een `if` statement waar we een achtergrondafbeelding of een andere achtergrond tekenden, gebaseerd op de waarde in de variabele `draw_background`. We doen hier hetzelfde, behalve in plaats van dat we alleen de afwisselende achtergrond weergeven, verhogen we de waarde in de variabele `count` door 1 toe te voegen (we gebruiken `count = count + 1`) bij ❹. Op basis van de waarde bij `count`, beslissen we welke afbeelding we tekenen. Bij ❺, als de waarde 5 heeft bereikt, tekenen we de plank (shelf) bij ❻. Bij ❼, als we de waarde 9 hebben bereikt, tekenen we de lamp ❽. Anders zouden we alleen de afwisselende achtergrond tekenen, zoals we eerder al gedaan hebben.

HOOFDSTUK 18

#1: "JIJ WINT!"

We kunnen de tekst "Jij wint!" toevoegen als een variabele in de `Spel` klasse in de `__init__` functie:

```
for x in range(0, 5):
    for y in range(0, 5):
        self.canvas.create_image(x * w, y * h, \
            image=self.bg, anchor='nw')
    self.sprites = []
    self.running = True
    self.game_over_text = self.canvas.create_text(250, 250, \
text='Jij wint!', state='hidden')
```

Om de tekst weer geven als het spel stop, hoeven we alleen een `else` statement in de `mainloop` functie toe te voegen:

```
    def mainloop(self):
        while 1:
            if self.running == True:
                for sprite in self.sprites:
                    sprite.move()
❶           else:
❷               time.sleep(1)
❸               self.canvas.itemconfig(self.game_over_text, \
                state='normal')
            self.tk.update_idletasks()
            self.tk.update()
            time.sleep(0.01)
```

Je kunt deze verandering zien in de regels ❶ tot ❸. We voegen een `else` voorwaarde toe aan het `if` statement bij ❶ en dan zal Python dit blok code uitvoeren als de variabele `running` niet meer op `True` staat.

Bij ❷ vertragen we een seconde zodat de tekst “Jij wint!” niet direct verschijnt, en vervolgens veranderen we de status van de tekst in `'normal'` bij ❸ zodat het op het doek verschijnt.

#2: DE DEUR ANIMEREN

Om ervoor te zorgen dat dat de deur opent als Mr. Stick Man de deur bereikt, wijzigen we de klasse `DoorSprite` als eerste. In plaats van de afbeelding als een parameter door te geven, zal de sprite nu twee afbeeldingen van de deur laden in de `__init__` functie:

```
class DoorSprite(Sprite):
    def __init__(self, game, x, y, width, height):
        Sprite.__init__(self, game)
❶       self.closed_door = PhotoImage(file="deur1.gif")
❷       self.open_door = PhotoImage(file="deur2.gif")
        self.image = game.canvas.create_image(x, y, \
            image=self.closed_door, anchor='nw')
        self.coordinates = Coords(x, y, x + (width / 2), y + height)
        self.endgame = True
```

Zoals je ziet, zijn de twee afbeeldingen als object variabele toegevoegd bij ❶ en ❷. We moeten de onderkant van de code van het spel veranderen waar we het `door` object hebben gemaakt zodat het niet langer probeert om de afbeelding parameter te gebruiken:

```
door = DoorSprite(g, 45, 30, 40, 35)
```

`DoorSprite` heeft twee nieuwe functies nodig. Eentje om de open deur afbeelding te tonen en eentje om de afbeelding van de gesloten deur te tonen.

```
    def opendoor(self):
❶       self.game.canvas.itemconfig(self.image, image=self.open_door)
❷       self.game.tk.update_idletasks()

    def closedoor(self):
❸       self.game.canvas.itemconfig(self.image,\
                image=self.closed_door)
        self.game.tk.update_idletasks()
```

Door de `itemconfig` functie van het doek te gebruiken, wijzigen we de weergegeven afbeelding in de afbeelding met de open deur die is opgeslagen in de `open_door` object variabele bij ❶. We roepen de `update_idletasks` functie van het `tk` object aan om ervoor te zorgen dat de nieuwe afbeelding wordt weergegeven bij ❷. (Als we dit niet doen, zal de afbeelding niet direct veranderen.) De `closedoor` functie is hetzelfde, maar toont de afbeelding die is opgeslagen in de `closed_door` variabele bij ❸.

De volgende nieuwe functie wordt toegevoegd aan de klasse `StickFigureSprite`:

```
    def end(self, sprite):
❶       self.game.running = False
❷       sprite.opendoor()
❸       time.sleep(1)
❹       self.game.canvas.itemconfig(self.image, state='hidden')
❺       sprite.closedoor()
```

We zetten de `running` object variabele van het spel op `False` bij ❶ en roepen de `opendoor` functie aan van de `sprite` parameter bij ❷. Dit is eigenlijk een `DoorSprite` object, welke we zien in het volgende deel van de code. Bij ❸ vertragen we 1 seconde, voordat we Mr. Stick Man laten verdwijnen bij ❹ en roepen dan de `closedoor` functie aan bij ❺. Dit laat

het erop lijken dat Mr. Stick Man door de deur is gegaan en deze achter zich heeft gesloten. De laatste aanpassing is de `move` functie van de `StickFigureSprite`.

In de eerdere versie van de code, toen Mr. Stick Man nog tegen de deur aanbotste, hebben we de `running` variabele op `False` gezet. Maar nadat we dit naar de `end` functie verplaatst hebben, moeten we die functie in plaats daarvan aanroepen:

```
            if left and self.x < 0 and collided_left(co, sprite_co):
                self.x = 0
                left = False
❶               if sprite.endgame:
❷                   self.end(sprite)
            if right and self.x > 0 and collided_right(co, sprite_co):
                self.x = 0
                right = False
❸               if sprite.endgame:
❹                   self.end(sprite)
```

In dit deel van de code waarin we nagaan of Mr. Stick Man naar links gaat en of hij aan de linkerkant tegen een sprite is aangebotst, controleren we of de `endgame` variabele op `True` staat bij ❶. Als dat zo is weten we dat het een `DoorSprite` object is en roepen we bij ❷ de `end` functie aan door gebruik te maken van de `sprite` variabele als parameter. We maken dezelfde wijziging in het deel van de code waar we zien of of Mr. Stick Man rechts gaat en aan de rechterkant tegen een sprite is aangebotst (bij ❸ en ❹).

#3: BEWEGENDE PLATFORMS

Een nieuwe klasse voor een bewegend platform zal er hetzelfde uitzien als de klasse voor Mr. Stick Man. We moeten de positie van het platform opnieuw berekenen, in plaats van het gebruik van bepaalde coördinaten. We kunnen een kind klasse van de `PlatformSprite` klasse maken. Dan wordt de `__init__` functie als volgt:

```
class MovingPlatformSprite(PlatformSprite):
❶   def __init__(self, game, photo_image, x, y, width, height):
❷       PlatformSprite.__init__(self, game, photo_image, x, y, \
          width, height)
❸       self.x = 2
❹       self.counter = 0
❺       self.last_time = time.time()
❻       self.width = width
❼       self.height = height
```

We doorlopen dezelfde parameters als bij de klasse `PlatformSprite` bij ❶, en roepen dan de `__init__` functie van de ouder klasse aan met dezelfde parameters bij ❷.

Dit betekent dat ieder object van de klasse `MovingPlatformSprite` precies hetzelfde wordt ingesteld als een object van de `PlatformSprite` klasse.

We kunnen een variabele x met een waarde van 2 maken (het platform start met naar rechts bewegen) bij ❸, gevolgd door een `counter` variabele bij ❹. We gebruiken deze om aan te geven wanneer het platform van richting moet veranderen. Omdat we niet willen dat het platform zo snel mogelijk heen en weer beweegt, net zoals de `StickFigureSprite` niet zo snel heen en weer mag bewegen, houden we de tijd bij in de `last_time` variabele bij ❺ (dit is de `last_time` variabele die we gebruiken om het platform te laten vertragen). De laatste toevoegingen bij deze functie zijn het opslaan van `width` en `height` bij ❻ en ❼.
De volgende toevoeging in de nieuwe klasse is de `coords` functie:

```
        self.last_time = time.time()
        self.width = width
        self.height = height

    def coords(self):
        xy = self.game.canvas.coords(self.image)
        self.coordinates.x1 = xy[0]
        self.coordinates.y1 = xy[1]
❶       self.coordinates.x2 = xy[0] + self.width
❷       self.coordinates.y2 = xy[1] + self.height
        return self.coordinates
```

De `coords` functie is bijna hetzelfde als die we hebben gebruikt voor Mr. Stick Man, behalve dat we nu de waarden gebruiken die we hebben opgeslagen in de `__init__` functie, in plaats van vaste waarden voor de breedte en hoogte. (Je kunt het verschil zien in de regels ❶ en ❷.) Omdat dit een bewegende sprite is, moeten we ook een `move` functie toevoegen:

```
        self.coordinates.x2 = xy[0] + self.width
        self.coordinates.y2 = xy[1] + self.height
        return self.coordinates
```

```
        def move(self):
❶           if time.time() - self.last_time > 0.03:
❷               self.last_time = time.time()
❸               self.game.canvas.move(self.image, self.x, 0)
❹               self.counter = self.counter + 1
❺               if self.counter > 20:
❻                   self.x = self.x * -1
❼                   self.counter = 0
```

De `move` functie gaat na of de tijd groter is dan 0.03 seconden bij ❶. Als dat zo is, zetten we de `last_time` variabele op de huidige tijd bij ❷. Bij ❸ bewegen we de afbeelding van het platform, en daarna hogen we de `counter` variabele op bij ❹. Als deze groter is dan 20 (de `if` statement bij ❺), draaien we van richting door de x variabele te vermenigvuldigen met -1 (als het positief is wordt het negatief, en als het negatief is wordt het positief) bij ❻, en resetten de counter tot 0 bij ❼. Nu zal het platform een richting opgaan voor een aantal van 20, en dan terug, de andere kant op, met een aantal van 20.

Om de bewegende platformen te testen, kunnen we een paar van de al bestaande platform objecten van `PlatformSprite` in `MovingPlatformSprite` veranderen:

```
platform5 = MovingPlatformSprite(g, PhotoImage(file="platform2.gif"), \
        175, 350, 66, 10)
platform6 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        50, 300, 66, 10)
platform7 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        170, 120, 66, 10)
platform8 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        45, 60, 66, 10)
platform9 = MovingPlatformSprite(g, PhotoImage(file="platform3.gif"), \
        170, 250, 32, 10)
platform10 = PlatformSprite(g, PhotoImage(file="platform3.gif"), \
        230, 200, 32, 10)
```

Hierna zie je de volledige code met alle veranderingen.

```
from tkinter import *
import random
import time

class Spel:
    def __init__(self):
        self.tk = Tk()
        self.tk.title("Mr. Stick Man holt naar de uitgang")
        self.tk.resizable(0, 0)
        self.tk.wm_attributes("-topmost", 1)
        self.canvas = Canvas(self.tk, width=500, height=500, \
                highlightthickness=0)
```

```
        self.canvas.pack()
        self.tk.update()
        self.canvas_height = 500
        self.canvas_width = 500
        self.bg = PhotoImage(file="achtergrond.gif")

        w = self.bg.width() h =
        self.bg.height()
            for x in range(0, 5):
                for y in range(0, 5):
                self.canvas.create_image(x * w, y * h, \
                            image=self.bg, anchor='nw')
        self.sprites = []
        self.running = True

        self.game_over_text = self.canvas.create_text(250, 250, \
                    text='Jij wint!', state='hidden')

    def hoofdlus(self):
        while 1:
            if self.running:
                for sprite in self.sprites:
                    sprite.move()
            else:
                time.sleep(1)
                self.canvas.itemconfig(self.game_over_text, \
                            state='normal')
            self.tk.update_idletasks()
            self.tk.update() time.sleep(0.01)

    class Coords:
        def __init__(self, x1=0, y1=0, x2=0, y2=0):
            self.x1 = x1
            self.y1 = y1
            self.x2 = x2
            self.y2 = y2

    def within_x(co1, co2):
        if (co1.x1 > co2.x1 and co1.x1 < co2.x2) \
                or (co1.x2 > co2.x1 and co1.x2 < co2.x2) \
                or (co2.x1 > co1.x1 and co2.x1 < co1.x2) \
                or (co2.x2 > co1.x1 and co2.x2 < co1.x2):
            return True
        else:
            return False

    def within_y(co1, co2):
        if (co1.y1 > co2.y1 and co1.y1 < co2.y2) \
                or (co1.y2 > co2.y1 and co1.y2 < co2.y2) \
                or (co2.y1 > co1.y1 and co2.y1 < co1.y2) \
                or (co2.y2 > co1.y1 and co2.y2 < co1.y2):
```

```
                return True
            else:
                return False

        def collided_left(co1, co2):
            if within_y(co1, co2):
                if co1.x1 <= co2.x2 and co1.x1 >= co2.x1:
                    return True
            return False

        def collided_right(co1, co2):
            if within_y(co1, co2):
                if co1.x2 >= co2.x1 and co1.x2 <= co2.x2:
                    return True
            return False

        def collided_top(co1, co2):
            if within_x(co1, co2):
            if co1.y1 <= co2.y2 and co1.y1 >= co2.y1:
                return True
            return False

        def collided_bottom(y, co1, co2):
            if within_x(co1, co2):
                y_calc = co1.y2 + y
                if y_calc >= co2.y1 and y_calc <= co2.y2:
                    return True
            return False

    class Sprite:
        def __init__(self, game):
            self.game = game
            self.endgame = False
            self.coordinates = None
        def move(self):
            pass
        def coords(self):
            return self.coordinates

    class PlatformSprite(Sprite):
        def __init__(self, game, photo_image, x, y, width, height):
            Sprite.__init__(self, game)
            self.photo_image = photo_image
            self.image = game.canvas.create_image(x, y, \
                    image=self.photo_image, anchor='nw')
            self.coordinates = Coords(x, y, x + width, y + height)
```

```
class MovingPlatformSprite(PlatformSprite):
    def __init__(self, game, photo_image, x, y, width, height):
        PlatformSprite.__init__(self, game, photo_image, x, y, \
                width, height)
        self.x = 2
        self.counter = 0
        self.last_time =
        time.time()
        self.width = width
        self.height = height

    def coords(self):
        xy = list(self.game.canvas.coords(self.image))
        self.coordinates.x1 = xy[0]
        self.coordinates.y1 = xy[1]
        self.coordinates.x2 = xy[0] + self.width
        self.coordinates.y2 = xy[1] + self.height
        return self.coordinates

    def move(self):
        if time.time() - self.last_time > 0.03:
            self.last_time = time.time()
            self.game.canvas.move(self.image, self.x, 0)
            self.counter += 1
            if self.counter > 20:
              self.x *= -1
              self.counter = 0

class DoorSprite(Sprite):
    def __init__(self, game, x, y, width, height):
        Sprite.__init__(self, game)
        self.closed_door = PhotoImage(file="deur1.gif")
        self.open_door = PhotoImage(file="deur2.gif")
        self.image = game.canvas.create_image(x, y, \
                image=self.closed_door, anchor='nw')
        self.coordinates = Coords(x, y, x + (width / 2), y + \
        height)
        self.endgame = True

    def opendoor(self):
        self.game.canvas.itemconfig(self.image,
        image=self.open_door) self.game.tk.update_idletasks()

    def closedoor(self):
        self.game.canvas.itemconfig(self.image, \
                image=self.closed_door)
        self.game.tk.update_idletasks()
```

```
class StickFigureSprite(Sprite):
    def __init__(self, game):
        Sprite.__init__(self, game)
        self.images_left = [
            PhotoImage(file="figuur-l1.gif"),
            PhotoImage(file="figuur-l2.gif"),
            PhotoImage(file="figuur-l3.gif")
        ]
        self.images_right = [
            PhotoImage(file="figuur-R1.gif"),
            PhotoImage(file="figuur-R2.gif"),
            PhotoImage(file="figuur-R3.gif")
        ]
        self.image = game.canvas.create_image(200, 470, \
                image=self.images_left[0], anchor='nw')
        self.x = -2
        self.y = 0
        self.current_image = 0
        self.current_image_add = 1
        self.jump_count = 0
        self.last_time = time.time()
        self.coordinates = Coords()
        game.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        game.canvas.bind_all('<KeyPress-Right>', self.turn_right)
        game.canvas.bind_all('<space>', self.jump)

    def turn_left(self, evt):
        if self.y == 0:
            self.x = -2

    def turn_right(self, evt):
        if self.y == 0:
            self.x = 2

    def jump(self, evt):
        if self.y == 0:
            self.y = -4
            self.jump_count = 0

    def animate(self):
        if self.x != 0 and self.y == 0:
            if time.time() - self.last_time > 0.1:
                self.last_time = time.time()
                self.current_image += self.current_image_add
                if self.current_image >= 2:
                    self.current_image_add = -1
                if self.current_image <= 0:
                    self.current_image_add = 1
         if self.x < 0:
            if self.y != 0:
                self.game.canvas.itemconfig(self.image, \
```

```
                image=self.images_left[2])
        else:
            self.game.canvas.itemconfig(self.image, \
                    image=self.images_left[self.current_image])

    elif self.x > 0:
        if self.y != 0:
            self.game.canvas.itemconfig(self.image, \
                image=self.images_right[2])
        else:
            self.game.canvas.itemconfig(self.image, \
                image=self.images_right[self.current_image])

    def coords(self):
        xy = list self.game.canvas.coords(self.image)
        self.coordinates.x1 = xy[0]
        self.coordinates.y1 = xy[1]
        self.coordinates.x2 = xy[0] + 27
        self.coordinates.y2 = xy[1] + 30
        return self.coordinates


    def move(self):
        self.animate()
        if self.y < 0:
            self.jump_count += 1
            if self.jump_count > 20:
                self.y = 4
        if self.y > 0:
            self.jump_count -= 1
        co = self.coords()
        left = True
        right = True
        top = True
        bottom = True
        falling = True
        if self.y > 0 and co.y2 >= self.game.canvas_height:
            self.y = 0
            bottom = False
        elif self.y < 0 and co.y1 <= 0:
            self.y = 0
            top = False
        if self.x > 0 and co.x2 >= self.game.canvas_width:
            self.x = 0
            right = False
        elif self.x < 0 and co.x1 <= 0:
            self.x = 0
            left = False
        for sprite in self.game.sprites:
            if sprite == self:
                continue
        sprite_co = sprite.coords()
```

```
                if top and self.y < 0 and collided_top(co, sprite_co):
                    self.y = -self.y
                    top = False
                if bottom and self.y > 0 and collided_bottom(self.y, \
                    co, sprite_co):
                    self.y = sprite_co.y1 - co.y2
                    if self.y < 0:
                        self.y = 0
                    bottom = False
                    top = False
                if bottom and falling and self.y == 0 \
                      and co.y2 < self.game.canvas_height \
                      and collided_bottom(1, co, sprite_co):
                    falling = False
                if left and self.x < 0 and collided_left(co, sprite_co):
                    self.x = 0
                    left = False
                    if sprite.endgame:
                        self.end(sprite)

                if right and self.x > 0 \
                        and collided_right(co, sprite_co):
                    self.x = 0
                    right = False
                    if sprite.endgame:
                        self.end(sprite)

            if falling and bottom and self.y == 0 \
                    and co.y2 < self.game.canvas_height:
                self.y = 4

            self.game.canvas.move(self.image, self.x, self.y)

        def end(self, sprite):
            self.game.running = False
            sprite.opendoor()
            time.sleep(1)
            self.game.canvas.itemconfig(self.image, state='hidden')
            sprite.closedoor()
```

```
g = Spel()
platform1 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
        0, 480, 100, 10)
platform2 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
        150, 440, 100, 10)
platform3 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
        300, 400, 100, 10)
platform4 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
        300, 160, 100, 10)
platform5 = MovingPlatformSprite(g, \
        PhotoImage(file="platform2.gif"), 175, 350, 66, 10)
platform6 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        50, 300, 66, 10)
platform7 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        170, 120, 66, 10)
platform8 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        45, 60, 66, 10)
platform9 = MovingPlatformSprite(g, \
        PhotoImage(file="platform3.gif"), 170, 250, 32, 10)
platform10 = PlatformSprite(g, PhotoImage(file="platform3.gif"), \
        230, 200, 32, 10)
g.sprites.append(platform1)
g.sprites.append(platform2)
g.sprites.append(platform3)
g.sprites.append(platform4)
g.sprites.append(platform5)
g.sprites.append(platform6)
g.sprites.append(platform7)
g.sprites.append(platform8)
g.sprites.append(platform9)
g.sprites.append(platform10)
door = DoorSprite(g, 45, 30, 40, 35)
g.sprites.append(door)
sf = StickFigureSprite(g)
g.sprites.append(sf)
g.hoofdlus()
```