Practicum opgave: simulatie van auto-verkeer



Deze opgave is geïnspireerd door een project van de EAVISE onderzoeksgroep, waarin beelden van verkeersstromen—zoals hieronder te zien—geanalyseerd worden om de verkeersveiligheid te vergroten. Zo kan bijvoorbeeld het effect van verschillende soorten wegmarkeringen in de buurt van zebrapaden op de snelheid van passerende auto's geanalyseerd worden.



A.1 Een eenvoudig datamodel: Auto

Om bij te houden welke auto's er op welke plaats in het beeld te zien zijn, maken we een eenvoudige klasse `Auto`.

Een auto heeft een bepaalde kleur, bevindt zich op een bepaalde (x, y) coördinaat in het vlak en heeft een lengte en een breedte. We stellen een auto dus gewoon voor als een rechthoek.

Oef. 1.1 ▸ Maak een klasse `Auto`. Voorzie een constructor waarmee je een auto kan aanmaken met zijn verschillende eigenschappen (kleur, x-positie, y-positie, lengte, breedte). De (x, y)-coördinaat van een auto is die van

zijn linkerbenedenhoek (dwz. zijn hoekpunt met de kleinste x en de kleinste y coördinaat).

Oef. 1.2 ▸ Schrijf onderaan in je bestand een `main` functie:

```
def main():
    ...
main()
```

Maak in deze functie een nieuw `Auto` object `auto1` aan dat de kleur "red" heeft, zich op positie $(10,20)$ bevindt, 30 lang en 20 breed is.

Oef. 1.3 ▸ Maak een tweede `Auto` object `auto2` aan met dezelfde gegevens als `auto1` en test wat er gebeurt met volgend stukje code:

```
if auto1 == auto2:
    print "gelijk"
else:
    print "verschillend"
```

Wat gebeurt er als je in plaats hiervan `auto1` met zichzelf vergelijkt?

Oef. 1.4 ▸ Schrijf in je klasse `Auto` een `__str__` methode die ervoor zorgt dat bijvoorbeeld `auto1` als volgt wordt afgeprint:

```
auto op (10,20) van grootte (30,20) in kleur red
```

Test uit of je methode effectief goed werkt.

Oef. 1.5 ▸ Schrijf in je klasse `Auto` een methode `omvat` waarmee je kan nagaan of een bepaalde (x,y)-positie door de auto bedekt wordt. Deze methode moet dus een boolean (`True` of `False`) teruggeven. Later gaan we deze methode gebruiken om bv. botsingen te detecteren. Schrijf in je `main` functie een test die het woord "botsing" afprint als `auto1` het punt $(20,30)$ bevat. Test dit ook eens uit met het punt $(0,0)$.

A.2 Een eenvoudig datamodel: Punt

In plaats van x en y coördinaten altijd apart mee te geven, kunnen we deze twee coördinaten ook samen nemen in één object van een nieuwe klasse `Punt`. Dit levert een netter resultaat op, met betere encapsulatie. Bovendien kunnen we deze klasse `Punt` dan ook gemakkelijk herbruiken.

Oef. 2.1 ▸ Maak een klasse `Punt` met een constructor die een x en y waarde als argument krijgt. Zorgt ervoor dat een punt wordt afgeprint in het formaat (x,y). Maak in je `main` functie het punt $(10,20)$ aan en controleer dat dit juist wordt afgeprint.

Oef. 2.2 ▸ Pas de constructor van de klasse `Auto` aan zodat er nu een `Punt` object wordt meegegeven als argument in plaats van twee aparte coördinaten. Pas ook de `__str__` methode aan en het aanmaken van de `Auto` objecten in je `main` functie. Controleer dat het afprinten van `auto1` nog steeds goed werkt. (Je `if`-test met de methode `omvat` zal nu niet meer werken. Zet deze even in commentaar.)

Oef. 2.3 ▸ Vervang ook het apart doorgeven van de lengte en de breedte in de constructor van `Auto` door het gebruik van één `Punt` en pas de `__str__` methode en je `main` functie ook hieraan aan. Controleer weer dat het afprinten nog goed werkt.

Oef. 2.4 ▸ Neem nu ook de methode `omvat` onder handen: vervang de twee afzonderlijke argumenten door één punt, zorg dat de methode weer werkt, en test dit uit in je `main` functie.

Oef. 2.5 ▸ Geef de klasse `Punt` een methode `afstand(zelf, ander)` waarmee je de afstand van een punt tot een ander punt kan bereken:

$$\delta((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Herinner je dat je eerst een `import math` moet doen, vooraleer je in Python de vierkantswortel functie kan gebruiken. Gebruik deze methode om in je `main`-functie te berekenen wat het verschil in afstand is tussen de linkerbenedenhoek van een auto op positie $(10, 20)$ en die van eentje op $(40, 60)$.

A.3 Een eenvoudig datamodel: Ruimte

Uiteindelijk willen we beelden kunnen analyseren waarop er verschillende auto's samen voorkomen. Hiervoor introduceren we een klasse `Ruimte`, die aggregatie is van `Auto`'s.

Oef. 3.1 ▸ Maak een klasse `Ruimte` met een constructor die een lege ruimte aanmaakt. Deze ruimte houdt een (initieel lege) lijst bij met alle auto's die erin aanwezig zijn. Geef deze klasse een methode `voegAutoToe` waarmee een auto aan de ruimte kan worden toegevoegd. Maak in je `main` functie twee auto's aan en gebruik deze methode om ze te voegen aan een ruimte r.

Oef. 3.2 ▸ Geef de klasse `Ruimte` een methode `__str__(zelf)`, die een string-voorstelling van de ruimte construeert, door de string-voorstellingen van alle auto's in deze ruimte te combineren. Controleer dat de string voorstelling van r inderdaad correct is.

We gaan nu detecteren of twee auto's in dezelfde ruimte al dan niet met elkaar botsen. Hiervoor moeten we aanpassingen doen aan de drie klassen die we al hebben.

Oef. 3.3 ▸ Schrijf in de klasse `Punt` een methode die twee punten bij elkaar optelt. Concreet moet deze methode een nieuw punt teruggeven dat het resultaat is van de optelling van het huidige punt (`zelf`) met een ander punt, dat als argument van de methode wordt meegegeven. Als je deze methode bovendien de naam `__add__` geeft, dan kan je deze oproepen door gewoon het symbool `+` te gebruiken (als in: `som = p1 + p2`, waarbij `p1` en `p2` twee `Punt` objecten zijn).

Oef. 3.4 ▸ Schrijf in de klasse `Auto` een methode `hoekpunten` die een lijst teruggeeft van vier `Punt`-objecten, die de vier hoekpunten van deze auto voorstellen.

Oef. 3.5 ▸ Schrijf in de klasse `Auto` een methode `botstMet` die nagaat of deze auto botst met een andere auto, door voor elk van zijn eigen hoekpunten aan de andere auto te vragen of deze dit hoekpunt omvat.

Oef. 3.6 ▸ Schrijf in de klasse `Ruimte` een methode `botsing` die nagaat of er in deze ruimte al dan niet twee *verschillende* auto's met elkaar in botsing zijn (elke auto is natuurlijk "in botsing met" zichzelf).

Tot slot beschouwen we nog een uitbreiding aan de klasse `Auto` die de "risico's" van gelijkheidstesten illustreert.

Oef. 3.7 ▸ Schrijf in de klasse `Auto` een methode `heeftAlsHoekpunt(punt)` die nagaat of een gegeven `Punt` object voorkomt in de lijst met hoekpunten van die auto. Gebruikt deze methode om te controleren of het punt $(40, 40)$ voorkomt in de lijst met hoekpunten van een auto op $(10, 20)$ van grootte $(30, 20)$. Mogelijk is het resultaat hier niet wat je verwacht. Ga eens na hoe dit komt.

Oef. 3.8 ▸ Zorg ervoor dat je methode `heeftAlsHoekpunt` wel goed werkt. Hiervoor moet je deze methode zelf niet aanpassen, maar wel een aanpassing doorvoeren aan de klasse `Punt`.

A.4 Visualisatie van een ruimte

We gaan nu onze ruimtes visualiseren met behulp van SVG prentjes in de webbrowser. Hiervoor overlopen we eerst alle verschillende stappen die nodig zijn om een Python programma te "installeren" op de webserver `laurel.local.thomasmore.be`. Daarna voegen we aan onze bestaande klassen code toe om de juist SVG tekst te produceren.

A.4.1 Installatie op de server

Oef. 4.1 ▸ Download het bestand

`http://iiw.denayer.be/~jve/svg.py`

en plaats het dmv. WinSCP in de map `public_html` op de server. Dit bestand heeft de volgende inhoud:

```
#!/usr/local/bin/python
import jve

svg = """
  <svg xmlns="http://www.w3.org/2000/svg">
     <rect x="50" y="20" width="150" height="150" fill="blue" />
  </svg>
"""

def main():
    print jve.http(svg)

main()
```

De eerste lijn dient om de webserver te vertellen dat dit een programma is dat moet worden uitgevoerd in Python. Vergeet ook het `xmlns`-attribuut in het `<svg>`-element niet, aangezien dit nodig is om Firefox te vertellen dat je een SVG-tekening wilt maken. Verder gebruiken we ook nog de functie `http` uit de module `jve` die we bovenaan importeren. Het effect hiervan wordt verderop duidelijk.

Oef. 4.2 ▸ Vertel de webserver dat hij dit programma mag uitvoeren. Dit doe je bijvoorbeeld door in WinSCP een terminal op de server te openen en daarin volgende commando's uit te voeren (het eerste is niet nodig als je terminal zich al in deze map bevindt):

```
% cd public_html
% chmod a+rx svg.py
```

Oef. 4.3 ▸ Je kan nu je programma als volgt uitvoeren:

```
% ./svg.py
```

Het resultaat zou er zo moeten uitzien:

```
% ./svg.py
Content-type: text/xml

<svg xmlns="http://www.w3.org/2000/svg">
    <rect x="50" y="20" width="150" height="150" fill="blue" />
</svg>
```

Je ziet hier meteen wat het effect is van de functie `http` uit de module `jve`: deze heeft gewoon bovenaan een extra lijn toegevoegd, die Firefox vertelt wat voor soort inhoud er volgt.

Oef. 4.4 ▸ Als je nu in Firefox naar volgende URL surft:

```
http://laurel.local.thomasmore.be/~aXXX/svg.py
```

zou je een blauwe rechthoek moeten zien.

A.4.2 Genereren van SVG code

Nu we in staat zijn om SVG tekeningen te maken in de webbrowser, gaan we dit gebruiken om de toestand van een ruimte te visualiseren.

Oef. 4.5 ▸ Breid de klasse `Auto` uit met een methode `svg` die een `<rect ... />` tag teruggeeft waarmee de auto wordt voorgesteld. Geef alle attributen van het `Auto` object hierbij hun juiste plaats.

Oef. 4.6 ▸ Breid de klasse `Ruimte` uit met een methode `svg` die een `<svg ...> ... </svg>` element teruggeeft met daarin de SVG voorstelling van alle auto's in deze klasse.

Oef. 4.7 ▸ Schrijf een `main` functie die een ruimte aanmaakt met twee auto's erin en die de SVG visualisatie hiervan toont (zoals je met `svg.py` gedaan hebt). Bekijk dat je in Firefox effectief het juiste prentje te zien krijgt (in SVG prentjes loopt de Y-as van boven naar beneden).
Tip: Als je in de webbrowser niet het gewenste resultaat ziet, probeer dan je programma uit te voeren in een terminal (met het commando `./svg.py`) . Zo zie je vaak sneller wat er mis is.

A.5 Overerving: bewegende auto's

Uiteindelijk willen we niet alleen maar een momentopname van de positie van verschillende auto's kunnen voorstellen, maar ook volgen hoe deze zich bewegen, zodat we bijvoorbeeld ook de snelheid van een auto kunnen berekenen. Hiervoor maken we een subklasse die de functionaliteit van `Auto` uitbreidt.

Oef. 5.1 ▸ Om te beginnen, geven we de klasse `Auto` een extra methode `verplaats`, waarmee we de positie van de auto kunnen aanpassen. Deze methode krijgt een nieuwe positie (een `Punt` object) als argument en vervangt de vorige positie van de auto door deze nieuwe.

Oef. 5.2 ▸ Maak een auto die begint op positie $(30, 40)$ en die zich daarna verplaatst naar $(50, 40)$. Print de auto voor en na de verplaatsing af om zeker te zijn dat alles correct is.

Nu zijn onze auto's misschien al wel verplaatsbaar, maar we zijn nog altijd niet in staat om bijvoorbeeld hun gemiddelde snelheid te berekenen, aangezien de auto's onmiddellijk vergeten wat hun vorige positie was. We gaan nu een nieuwe klasse maken, waarmee we auto's kunnen maken die hun vorige posities wel onthouden. Om geen code onnodig te moeten dupliceren, laten we deze klasse overerven van de bestaande klasse `Auto`.

Oef. 5.3 ▸ Maak een klasse `AutoMetGeschiedenis` die overerft van de klasse `Auto`. In deze klasse, moet je twee methodes van de oorspronkelijke klasse `Auto` overschrijven: (1) je maakt een nieuwe constructor, die hetzelfde doet als de constructor van `Auto` maar daarnaast ook nog een attribuut `geschiedenis` maakt met daarin een lijst die de huidige positie van de auto bevat; (2) je overschrijft de methode `verplaats`, zodat deze niet alleen de huidige positie vervangt door de nieuwe, maar ook de nieuwe positie toevoegt aan de geschiedenis-lijst.

Oef. 5.4 ▸ Pas je `main`-functie aan zodanig dat er nu een `AutoMetGeschiedenis` object wordt aangemaakt ipv. een gewone `Auto`. Natuurlijk zal je nog geen verschil zien in hoe dit object van deze nieuwe klasse zich gedraagt in je `main`-functie.

Oef. 5.5 ▸ Zorg ervoor dat een `AutoMetGeschiedenis` wordt afgeprint in het volgende formaat:

```
onthoudende auto op (10,20) van grootte (30,20) in kleur red
```

(Merk op dat dit, op het eerste woord na, identiek is aan de stringvoorstelling van een gewone auto.)

Oef. 5.6 ▸ Voeg aan je klasse `AutoMetGeschiedenis` een methode `snelheden` toe die een lijst teruggeeft met daarin de snelheden die de auto behaald heeft tussen de verschillende punten in zijn geschiedenis. Met andere woorden, als de geschiedenis van de auto bestaat uit punten

$$[p_0, p_1, \ldots, p_n]$$

dan produceert deze methode als resultaat de lijst

$$[d_0, d_1, \ldots, d_{n-1}]$$

waarbij elke $d_i = \delta(p_i, p_{i+1})$. Laat in je `main`-methode een auto rijden van $(0, 10)$ naar $(20, 10)$ naar $(50, 10)$ naar $(80, 10)$ en controleer of zijn lijst met snelheden inderdaad correct is.

Oef. 5.7 ▸ Voeg aan je klasse `AutoMetGeschiedenis` een methode `gemiddeldeSnelheid` toe, die de gemiddelde snelheid van de auto (dwz. de gemiddelde waarde van de lijst met snelheden) teruggeeft. Controleer of de gemiddelde snelheid van de auto uit de vorige oefening inderdaad correct is.

A.6 Visualisatie van oude toestanden

We hebben nu al een datastructuur waarmee we het traject dat een auto aflegt, kunnen bijhouden. Nu gaan we dit ook visualiseren.

Als eerste stap, maken we het mogelijk om de toestand van een auto (en bij uitbreiding ook een ruimte) op eender welk tijdstip te (her-)bekijken.

Oef. 6.1 ▸ Breidt de klasse `AutoMetGeschiedenis` uit met een methode `reset(i)` die de positie van de auto terugzet naar het i-de element uit zijn geschiedenis. Geef het argument van deze methode een *default*-waarde, zodat er naar het allereerste tijdstip 0 wordt teruggegaan als de methode wordt opgeroepen zonder argument.

Oef. 6.2 ▸ Geef de klasse `Ruimte` eenzelfde methode die elke auto uit de ruimte terugzet naar zijn i-de positie en voorzie ook hier een *default*-waarde 0.

Oef. 6.3 ▸ Maak in je `main`-functie een ruimte met twee auto's die elk vier posities hebben en controleer (door je ruimte te `reset`ten en af te printen) dat alles correct werkt.

We gaan nu de toestand op het i-de moment visualiseren met een SVG prentje.

Oef. 6.4 ▸ Maak in je `main`-functie nog steeds dezelfde ruimte met twee auto's aan en zorg dat er een SVG-prentje geproduceerd wordt dat de toestand van de ruimte op het 2^e tijdstip toont. Bekijk dit in Firefox. Controleer dat je ook voor andere tijdstippen het juiste resultaat krijgt, door het cijfer 2 te veranderen in iets anders.

In plaats van telkens onze code te moeten aanpassen als we een ander tijdstip willen zien, is het beter om hiervoor een webformulier te maken. Dit doen we nu in een aantal stappen.

Oef. 6.5 ▸ Zorg ervoor dat je Python-programma een query-string van de vorm `tijdstip=`i aanvaardt, en de waarde i doorgeeft als argument aan de `reset` functie. Het resultaat moet met andere woorden zijn dat als je surft naar volgende URL, je de toestand van de ruimte op het derde tijdstip te zien zal krijgen:

```
http://laurel.local.thomasmore.be/~aXXX/svg.py?tijdstip=3
```

Je kan hiervoor vertrekken van het voorbeeld in Sectie 4.1. Mogelijk heb je ook nood aan de functie `int(string)` die een string zoals `"7"` omzet in een getalwaarde.

Oef. 6.6 ▸ Maak een webpagina `auto.html` met daarop een formulier. In dit formulier, moet zich een tekstveldje bevinden, waarin een gebruiker het tijdstip dat hij wenst te zien, kan invullen. Het formulier moet daarnaast natuurlijk ook een `submit`-knop hebben. Als de gebruiker hierop klikt, dan wordt het getal dat hij heeft ingevuld doorgestuurd naar je Python-programma, zodat de gebruiker een afbeelding krijgt met daarop de toestand op het juiste tijdstip.

Ons programma rekent er nu nog op dat de gebruiker altijd “goede” invoer zal leveren. Dat is natuurlijk een beetje een naïeve veronderstelling.

Oef. 6.7 ▸ Bekijk eens in Firefox wat er gebeurt als je een getal ingeeft dat groter is dan het aantal posities van de auto’s. Bekijk ook eens (vanuit een terminal) wat er gebeurt als je vanuit je `main`-functie de `reset`-methode oproept met dit getal.

Oef. 6.8 ▸ Voeg aan de methode `reset` in de klasse `AutoMetGeschiedenis` een test toe die controleert of het opgegeven tijdstip wel degelijk in het juiste interval valt. Indien dit niet zo is, moet je een `IndexError` uitzondering `raise`n met de boodschap “Ongeldig tijdstip”. Bekijk in je editor of je deze uitzondering inderdaad ziet verschijnen.

Oef. 6.9 ▸ We gaan er nu voor zorgen dat als de gebruiker een verkeerd tijdstip ingeeft, hij gewoon de toestand op tijdstip 0 te zien krijgt. Om dit voor mekaar te krijgen, moet je de `IndexError` uitzondering opvangen op de juiste plaats in de klasse `Ruimte`.

Oef. 6.10 ▸ Controleer dat dit goed werkt. Bekijk ook eens wat er gebeurt als de gebruiker in het tekstveld iets zou invullen dat helemaal geen getal is. Hoe moet je je code aanpassen om ook in dit geval de toestand op tijdstip 0 te tonen? (Je hoeft hiervoor zelf geen nieuwe uitzondering op te steken, maar je kan gewoon de uitzondering die Python genereert opvangen. Eigenlijk hadden we dit ook al kunnen doen met de `IndexError`, maar dan had je natuurlijk het `raise`n van een uitzondering niet kunnen inoefenen.)

A.7 Dynamische visualisatie van een ruimte

> Nu we de verschillende toestanden van een ruimte individueel kunnen visualiseren, gaan we deze samennemen om hiermee een filmpje te maken.

De module `jve` bevat een functie `movie` die je hiervoor kan gebruiken. De volgende code toont bijvoorbeeld een filmpje van een blauwe bal die naar rechts rolt:

```
#!/usr/local/bin/python
import jve

svg = """
  <svg id="frame1" xmlns="http://www.w3.org/2000/svg">
      <circle cx="20" cy="20" r="10" fill="blue" />
  </svg>
  <svg id="frame2" xmlns="http://www.w3.org/2000/svg">
      <circle cx="30" cy="20" r="10" fill="blue" />
  </svg>
  <svg id="frame3" xmlns="http://www.w3.org/2000/svg">
      <circle cx="40" cy="20" r="10" fill="blue" />
  </svg>
"""

print jve.movie(svg)
```

Pas jouw programma aan zodat op dezelfde manier een filmpje getoond wordt:

Oef. 7.1 ▸ Vervang de instructie `jve.http(...)` door `jve.movie(...)`. Als je je programma nog eens uitvoert in een terminal, zal je zien dat deze functie `jve.movie` nog wat meer code rond jouw SVG tags geproduceerd heeft. Deze dient om ervoor te zorgen dat je verschillende SVG elementen als opeenvolgende frames van een filmpje kan afspelen. Daarvoor moeten deze wel "genummerd" zijn met de juiste `id`-attributen (zoals in het voorbeeld hierboven).

Oef. 7.2 ▸ Zorg ervoor dat de `svg`-methode in de klasse `Ruimte` een extra argument i aanvaardt, en dit gebruikt om aan het `<svg>` element dat het produceert een attribuut `id="frame`i`"` toe te voegen.

Oef. 7.3 ▸ Maak in de klasse `Ruimte` een bijkomende methode `alle_svgs(n)` die de SVG elementen voor tijdstippen 0 tot en met $n-1$ verzamelt, achter elkaar plakt en teruggeeft.

Oef. 7.4 ▸ Laat je `main`-methode `alle_svgs(4)` teruggeven en controleer dat je in Firefox effectief een filmpje te zien krijgt.

A.8 Regio's in het beeld

> Een interessante vraag om te bekijken, is of de snelheid van auto's in sommige regio's (bv. waar er een zebrapad is) hoger of lager ligt dan elders

Een regio zal een rechthoekig gebied zijn in de ruimte, dat we zullen voorstellen door een hoekpunt en een lengte–breedte paar. Dit komt je waarschijnlijk bekend voor, aangezien we auto's natuurlijk ook op deze manier voorspellen. Om duplicatie van code te vermijden, zullen we dan ook een hiërarchie van drie klassen maken: een superklasse `Rechthoek` met zowel een subklasse `Auto` als een subklasse `Regio`.

Oef. 8.1 ▸ Maak een klasse `Rechthoek`. Elke rechthoek bestaat uit (x, y)-coördinaten van de linkerbenedenhoek en een lengte–breedte paar. Er moet een constructor zijn om dergelijke rechthoeken mee aan te maken, en een methode die nagaat of een bepaald punt zich al dan niet in de rechthoek bevindt. We kunnen nu een auto zien als een speciaal soort rechthoek, namelijk één die zich kan verplaatsen en die ook een kleur heeft. Maak daarom van de klasse `Rechthoek` een superklasse van de klasse `Auto` en plaats de verschillende methodes in de juiste klasse.

Oef. 8.2 ▸ Maak nu een tweede subklasse `Regio` van `Rechthoek`. Een `Regio` heeft, naast de attributen van een rechthoek, ook nog een naam. Daarnaast moet een regio ook een methode `svg` hebben die de regio tekent als een `rect` met een `fill` van `none` en een `stroke` van `black`.

Oef. 8.3 ▸ Breid de klasse `Ruimte` uit, zodanig dat elke ruimte ook een lijst van regio's bijhoudt. Dit is een lege lijst op het moment dat het `Ruimte` object wordt aangemaakt. Er moet een methode `voegRegioToe` zijn om later regio's toe te voegen. Zorg ervoor dat de regio's eveneens getekend worden in de SVG voorstelling van een ruimte.

Oef. 8.4 ▸ Maak een ruimte met daarin een auto van 30 op 20 die van $(0, 10)$ naar $(20, 10)$ naar $(50, 10)$ naar $(80, 10)$ rijdt, en een even grote auto die van $(100, 100)$ naar $(90, 100)$ naar $(80, 100)$ naar $(70, 100)$ rijdt. Voeg nu ook een regio "bovenkant" toe op $(0, 0)$ met lengte 100 en breedte 60, en een tweede regio "onderkant" die even groot is maar op $(0, 60)$ begint. Bekijk de visualisatie van deze ruimte om na te gaan dat alles correct is.

Oef. 8.5 ▸ Om het onszelf niet te moeilijk te maken, gaan we ervan uit dat een auto **altijd binnen dezelfde regio blijft**. Schrijf in de klasse `Ruimte` een methode `gemiddeldeSnelheidVoorRegio(regio)` die de gemiddelde snelheid van alle auto's in de regio berekent. Hiervoor overloop je alle auto's en test dan voor elke auto of één van zijn hoekpunten in de regio valt. Zo ja, dan neem je de gemiddelde snelheid van die auto mee in de berekening; anders negeer je de auto.

Oef. 8.6 ▸ Voeg aan de klasse `Ruimte` een methode `analyseerRegios` toe, die alle regio's uit de ruimte overloopt en voor elke regio zijn naam afprint, gevolgd door de gemiddelde snelheid van de auto's in die regio.

Practicum opgave: data analyse



Deze opgave wordt elektronisch ter beschikking gesteld in de vorm van een iPython Notebook, waarin je meteen de oplossingen van de oefeningen kan invullen en laten uitvoeren. Dergelijke notebooks zijn immers ook de omgeving die in de praktijk het meest gebruikt wordt voor data analyse in Python. Je kan het dit notebook vinden in het bestand `BikeSharingOpgave.ipynb` dat je via Toledo kan downloaden.