Week 10 – Labo voor MCT

Doelstellingen

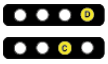
Je bent in staat om Object Georiënteerd Programmeren via de programmeertaal Python toe te passen. Concreet pas je volgende zaken toe:

- Ontwerp en programmeren van data-klassen waartussen overerving bestaat.
- Exception handling waar nodig
- File I/O & OOP: verwerken van json-bestanden
- Implementeren van test-methode

Dit labo-document is specifiek voor studenten MCT. Ben je MIT student? Dan werk je deze week aan andere oefeningen.

Afspraken

Eindniveau - oefeningen



Ben je een student MCT, dan beheers je de oefeningen tot moeilijkheidsgraad “D”
Ben je een student MIT, dan beheers je de oefeningen tot moeilijkheidsgraad “C”

GitHub

Alle oplossingen van week 10 dienen op Github geplaatst te worden. Volg hiervoor de procedure uitgelegd op Leho. Via Github krijg je ook alle bronmateriaal voor deze opgave.

Om je repository in onze Github-classroom aan te maken, klik je op volgende link:
https://classroom.github.com/a/LSj0b4bn (oef1)
https://classroom.github.com/a/s7R7iOnC (oef2)

Na elke oefening kan je een 'commit & push' doen zodat jouw versie op GitHub steeds aangepast wordt. Geef telkens een gepaste message mee.

Niet afgewerkte oefeningen werk je thuis verder af: voer regelmatig een 'push & commit'-opdracht uit zodat alle oplossingen op je github-repository beschikbaar zijn.

Bij een programmeertaal zoals Python onder de knie krijgen is veelvuldig oefenen essentieel en een noodzakelijke voorwaarde. Daarom vind je in elk labo-document nog twee extra onderdelen. Deze worden als volgt aangeduid.

Uitbreidingsoefeningen - eigen onderzoek

Dit onderdeel gaat verder dan de geziene leerstof van deze week. Vaak zijn de opdrachten net iets moeilijker dan hetgeen je in het labo deed. Je zal de Python documentation en Google nodig hebben voor dit onderzoek.

We motiveren iedereen om dit (thuis) iedere week voor te bereiden. Je onderzoekt in dit onderdeel een onderwerp die de volgende weken terugkomt in de theorie of het labo.

- Oefeningen voor thuis

In dit onderdeel vind je analoge oefeningen zoals je reeds in het labo maakte.
Deze oefeningen hebben dezelfde moeilijkheidsgraden als in het labo. Het is pas door de oefeningen thuis “alleen” te maken dat je je de leerstof eigen maakt. Loop je vast bij een oefening? Herbekijk de theorie, kijk of je een analoge oefening terugvindt die je maakte tijdens het labo. Lukt het nog steeds niet? Kom met je voorbereiding naar het monitoriaat!

Feedback labo week 9

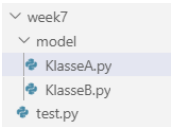
Lees nog even de algemene opmerkingen uit de theorieles na. Je vindt ze op Leho terug.

Officiële documentatie van Python: https://docs.python.org/3.9/
Handige tutorial: https://docs.python.org/3.9/tutorial/index.html
Naming conventions binnen Python: https://www.python.org/dev/peps/pep-0008/

Opgelet!

Vaak gebeuren er fouten tegen het importeren van een klasse.

Voorbeeld:
Klasse A maak gebruik van klasse B. Klasse A moet daarom klasse B importeren.
De testmethode zit één niveau hoger



Klasse A importeert klasse B.
Klasse A en klasse B zitten in dezelfde map.
We kijken vanuit het standpunt van de geopende projectmap!
	`from .KlasseB import KlasseB`
	`# alternatief`
	`# from .KlasseB import KlasseB`
	
	`class KlasseA:`
	
	`  def __init__(self):`
	`  self.__var1 = KlasseB()`

Test.py maakt gebruik van de klasseA.
We vermelden de submap ‘model’ waarin hij moet zoeken.
	`from model.KlassA import KlassA`
	
	`object_a1 = KlassA()`
	`object_a2 = KlassA()`

*Opm: wanneer jouw pylint actief is, dan geeft hij bij onderstaande methode een warning. Dit is evenwel **GEEN** uitvoeringsfout!*

```
1  from .KlasseB import KlasseB
2
3  class KlasseA:
4
5      def __init__(self):
6          self.__var1 = KlasseB()
```

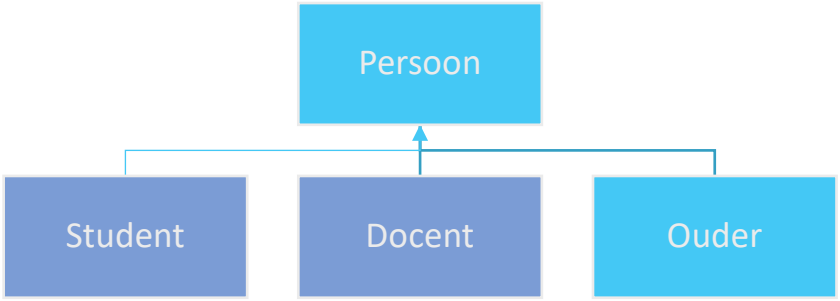
Oefeningen

Vermeld in commentaar telkens de opgave!

Oef 01

Deel 1

Werk verder op de demo uit de theorieles. (Zie ook bronmateriaal bij deze opgave)
Voeg de klasse '**Ouder**' toe die erft van de klasse Persoon.



Deze klasse houdt één extra attribuut bij, nl. een list van studenten (we gaan er voor de eenvoud vanuit dat alle kinderen studenten zijn). In de init-methode zet je hiervoor een lege list klaar. In deze list komen dus objecten van de klasse Student.

*Let op met het gebruik vande juiste termen: de klasse **Ouder** erft van de klasse **Persoon** over. De klasse **Ouder** houdt een **list van objecten** van de klasse **Student** bij: tussen de klasses Ouder en Student is er dus een associatie aanwezig.*

Werk deze klasse nu systematisch uit. Test regelmatig (via de beschikbare testcode).

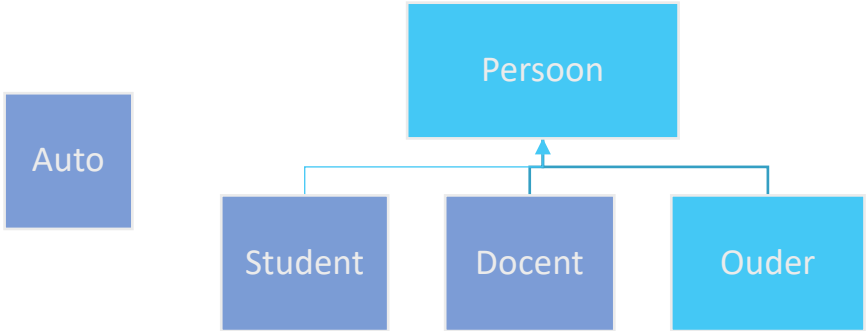
- Implementeer de methodes __init__( ) & __str__( )
- Een property-methode 'studenten' die de list van studenten teruggeeft.
  Een setter-methode hoeft niet toegevoegd te worden.
- Een methode '*voeg_student_toe*' langs waar een student kan toegevoegd worden. Controleer vooraf of de student nog niet in de list aanwezig is.
  Welke achterliggende methode uit de klasse Student wordt bij deze controle gebruikt?
- Een methode '*geef_info_studenten'* die een string teruggeeft waarin zowel de info van de ouder als de info over elke student verwerkt zit.
- Een methode '*geef_opleidingen_studenten*' die een list met de namen van de opleidingen van de kinderen teruggeeft. Dubbels mogen niet voorkomen.

Test alle methodes uit.

Maak finaal ook een nieuw object van de klasse Auto aan waarbij de ouder als eigenaar wordt ingesteld.

Hoe verhoudt de klasse Persoon zich tov de klasse Auto?

Print de __str__() van de klasse Auto af. Dient de klasse Auto aangepast te worden? Waarom wel/niet?



Opmerking: Je kan de test-methode in het bronmateriaal van deze oefening gebruiken.

Een voorbeeld kan er als volgt uitzien:

```
Ouder SPRIET, Marc (1951)

Ouder SPRIET, Marc (1951)  heeft volgende studenten:
Student SPRIET, Nick (1995) , 199510548 volgt de opleiding NMCT
Student SPRIET, Fien (1994) , 199510412 volgt de opleiding DAE
Student SPRIET, Marieke (1992) , 199212212 volgt de opleiding NMCT

De verschillende opleidingen van de kinderen zijn: ['NMCT', 'DAE']

Ouder wordt de eigenaar van een auto:
Voertuig met kenteken 1-DVX-656 heeft als eigenaar: Ouder SPRIET, Marc (1951)
```

Deel 2: inlezen van een json-file

1: Plaast het bestand 'personen.json' in de submap 'doc'.
Bestudeer aandachtig het json-bestand, controleer waar er [] en {} staat . Welke data vind je precies terug?

2: In de submap 'model' voeg je een extra klasse **PersoonRepository** toe om de json in te lezen en te filteren.

3: **PersoonRepository**:

- Voeg een public static methode '**inlezen_ouders_met_studenten**' toe die in staat is om het bronbestand in te lezen en een list van ouders met hun studenten terug te geven.
  Opgelet: tijdens het verwerken van de data van een ouder lees je ook de studenten mee in. Maak dus eerst een object van de klasse Ouder aan, waarna

er 0 of meerdere objecten van de klasse Student via de methode *voeg_student_toe* toegevoegd worden.

- Opgelet: je zal zien dat er geen studentnummer in de json file terug te vinden is.
  Geef een student dus een random studentennummer tussen 1 en 10000, als je een student aanmaakt.

Test de methode uit. Nadien print je van alle ingelezen ouders de verschillende opleidingen (geen dubbels) van hun kinderen af. Gebruik hiervoor de juiste methode uit de klasse Ouder.

test_wagens_zonder_json.py

```
def test_json():
    list_ouders = PersoonRepository.inlezen_ouders_met_studenten()
    for ouder in list_ouders:
        print(ouder.geef_info_studenten())

    for ouder in list_ouders:
        print(f"De kinderen van {ouder} volgen deze opleidingen: {ouder.geef_opleidingen_stude
nten()}")
```

Terminal

```
Ouder SPRIET, Marc (1951) heeft volgende studenten:
Student SPRIET, Nick (2019) volgt de opleiding MCT
Student SPRIET, Fien (2019) volgt de opleiding Verpleegkunde
Student SPRIET, Marieke (2019) volgt de opleiding MCT

Ouder VANNIEUWENHUYSE, Johan (1954) heeft volgende studenten:
Student VANNIEUWENHUYSE, Marieke (2019) volgt de opleiding Office management
Student VANNIEUWENHUYSE, Jan (2019) volgt de opleiding Burgerlijk ingenieur

Ouder DEHAENE, Lisa (1965) heeft volgende studenten:
Student DEHANE, Wout (2019) volgt de opleiding MIT

Ouder JANSENS, Wout (1978) heeft volgende studenten:

De kinderen van Ouder SPRIET, Marc (1951) volgen deze opleidingen: ['MCT', 'Verpleegkunde']
De kinderen van Ouder VANNIEUWENHUYSE, Johan (1954) volgen deze opleidingen: ['Office
management', 'Burgerlijk ingenieur']
De kinderen van Ouder DEHAENE, Lisa (1965) volgen deze opleidingen: ['MIT']
De kinderen van Ouder JANSENS, Wout (1978) volgen deze opleidingen: []
```

3: **Zoekopdracht**

Voeg de static methode '**filter_ouders_met_studenten_uit_opleiding**' toe aan de klasse PersoonRepository. Deze methode heeft als parameters de ingelezen list met ouders én een een opleidingsnaam. De ouders worden in de methode overlopen: enkel deze ouders waarvan één of meerdere studenten de doorgegeven opleiding volgen, worden in een list op het einde terug gegeven.

test_wagens_zonder_json.py

```
def test_json():
    list_ouders = PersoonRepository.inlezen_ouders_met_studenten()
    naam_opleiding = input("\nGeef een opleiding op:> ")
    list_resultaat = PersoonRepository.filter_ouders_met_studenten_uit_opleiding(list_ouders,
                     naam_opleiding)
    print("Resultaat zoekopdracht: ")
    for ouder in list_resultaat:
        print(f"{ouder} heeft student(en) die de opleiding {naam_opleiding} volgen.")
```

Terminal

```
Geef een opleiding op:> MIT
Resultaat zoekopdracht:
Ouder SPRIET Marc heeft student(en) die de opleiding MIT volgen.
Ouder DEHAENE Lisa heeft student(en) die de opleiding MIT volgen.
```

Oef 02

Gegeven volgende beschrijving:

Het taxibedrijf How@car heeft sinds kort, naast drie wagens ook twee lichte vrachtwagens rondrijden.

Van elk **voertuig** wordt standaard het *id*, *bouwjaar*, het *merk*, het *kmstand* en de *reisbestemming* bijgehouden.

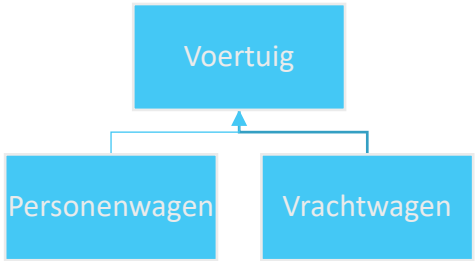
Voor de **personenwagens** wordt er extra het *maximum aantal passagierplaatsen* en het *aantal vervoerde personen* bijgehouden.

Analoog voor de lichte **vrachtwagens** wordt ook het *maximaal* toegelaten *laadvermogen* en het *gewicht* van de vervoerde *vracht* bijgehouden.

Vanuit een monitorcentrum wenst men een applicatie die de status van alle personenwagens kan weergeven en aanpassen.

Werkwijze - Dataklassen

Maak de dataklassen **Voertuig**, **Personenwagen** en **Vrachtwagen**.
Welke relatie bestaat er tussen deze klasses?



Klasse **Voertuig**

- Voorzie de klasse van de nodige private attributen:
  - *id, merk, bouwjaar, kmstand, reisbestemming*
- Maak voor elk attribuut een publieke property aan.
  - *id*, *merk* en *bouwjaar* heeft enkel een **getter** property
    - Een merk wordt altijd in hoofdletters teruggegeven.
  - *kmstand* heeft een **setter** en **getter** property
    - Voorzie inputvalidatie in de property-**setter**.
      Een kmstand moet een geheel- of kommagetal zijn.
  - reisbestemming heeft een **setter** en **getter** property
    - Voorzie inputvalidatie in de property-setter.
      Een reisbestemming moet een string zijn.
      - (een lege string is ook geldig, let er dus op dat deze aanvaard wordt).
- De constructor
  - heeft 4 parameters: *id, merk, bouwjaar, kmstand*

    - *reisbestemming* wordt ingesteld als lege string
- Welke **operators** moet je overwriten om later volgende functionaliteiten te voorzien?
    - Voertuigen worden gesorteerd op het aantal afgelegde kilometers.
    - 2 voertuigen zijn gelijk aan elkaar als het *id* gelijk is.
    - Je moet zowel een object van het type Voertuig, als een list van een voertuigen kunnen afprinten.
- Voorzie de klasse Voertuig van een static- attribuut '__aantal_voertuigen' (die het aantal gecreëerde objecten van deze klasse bijhoudt) en een static-methode 'geef_aantal_voertuigen' die het aantal teruggeeft.

Klasse **Vrachtwagen** erft van de klasse Voertuig

- Voorzie de klasse van 2 extra attributen:
    - *vracht*, *max_laadvermogen*
- Maak voor deze attributen een publieke propertie aan. Bij foutieve input geef je een ValueError terug
    - *max_laadvermogen* heeft een **setter** en **getter** property
        - Voorzie inputvalidatie in de property-setter.
          Het laadvermogen moet een geheel- of kommagetal zijn.
    - *vracht* heeft een **setter** en **getter** property
        - Voorzie inputvalidatie in de property-setter.
          Het gewicht van de vracht moet een geheel- of kommagetal zijn.
          Het gewicht van de vracht moet tevens kleiner zijn dat het *laad vermogen*
- De constructor
    - heeft de 5 parameters: *id, merk, bouwjaar, kmstand* en *max_laadvermogen*
    - *vracht* wordt ingesteld op 0
- Voorzie een gepaste ToString (str) methode.
- Heeft een extra methode '**geef_detail_info**'
    - Zie het codevoorbeeld voor de gewenste output.

Klasse **Personenwagen** erft van de klasse Voertuig

- Voorzie de klasse van 2 extra attributen:
    - *max_aantal_zitplaatsen*, *aantal_personen*
- Maak voor deze attributen een publieke propertie aan. Bij foutieve input geef je een ValueError terug
    - *max_aantal_zitplaatsen* heeft een **setter** en **getter** property
        - Voorzie inputvalidatie in de property-setter.
          Het max aantal zitplaatsen moet een geheel getal zijn én positief zijn
    - *aantal_personen* heeft een **setter** en **getter** property

        - Voorzie inputvalidatie in de property-setter.
          Het aantal personen moet een geheel getal zijn én kan niet groter zijn dan het *max_aantal_zitplaatsen*.
- Maak een **extra getter**-property (zonder attribuut) *vrije_plaatsen*
    - Deze property geeft het aantal vrije plaatsen weer in het voertuig. Hoe bereken je dit?
- De constructor
    - heeft de 5 parameters: *id, merk, bouwjaar, kmstand* en *max_aantal_zitplaatsen*
    - *aantal personen* aan boord wordt ingesteld op 0
- Voorzie een gepaste ToString methode.
- Heeft een extra methode '**geef_detail_info**'
    - Zie het codevoorbeeld voor de gewenste output.

Test uit door verschillende objecten van deze klasse aan te maken.
De output kan er als volgt uitzien:

test_wagens_zonder_json.py

```
from model.Personenwagen import Personenwagen
from model.Vrachtwagen import Vrachtwagen

rental1 = Personenwagen(100, "Audi", 2010, 3000, 5)
rental2 = Personenwagen(101, "Volkswagen", 2016, 0, 5)
rental3 = Vrachtwagen(102, "DAF", 2009, 3000, 5000)

list_met_rentals = [rental1, rental2, rental3]
print("*** Dit zijn de beschikbare voertuigen ****")
for item in list_met_rentals:
    print(item)

print("*** Geef de nieuwe bestemming van de voertuigen ****")
for item in list_met_rentals:
    temp_bestemming = input(f"Voor {item} > ")
    item.bestemming = temp_bestemming

print("*** Er stappen 3 personen in de Volkswagen ****")
list_met_rentals[1].aantal_personen = 3
print("*** De vrachtwagen wordt geladen ****")
try:
    temp_vracht = float(input(
        f"Hoeveel vracht neemt deze vrachtwagen mee? (max:
{list_met_rentals[2].max_laadvermogen}) > "))
    list_met_rentals[2].vracht = temp_vracht
except Exception as ex:
    print(f"Foutje: {ex}")

print("*** toon details per rental car ****")
for item in list_met_rentals:
    item.geef_detail_info()
```

Terminal

```
*** Dit zijn de beschikbare voertuigen ****
Personenwagen AUDI 2010
Personenwagen VOLKSWAGEN 2016
Vrachtwagen DAF 2009
*** Geef de nieuwe bestemming van de voertuigen ****
Voor Personenwagen AUDI 2010 > Brugge
Voor Personenwagen VOLKSWAGEN 2016 > Kortrijk
Voor Vrachtwagen DAF 2009 > Gent
*** Er stappen 3 personen in de Volkswagen ****
*** De vrachtwagen wordt geladen ****
Hoeveel vracht neemt deze vrachtwagen mee? (max: 5000)> 4000
```

```
*** toon details per rental car ****
Detailinfo over AUDI (2010):
        - ID: 100
        - Reisbestemming: Brugge
        - Aantal zitplaatsen: 5
        - Aantal vrije plaatsen: 5

Detailinfo over VOLKSWAGEN (2016):
        - ID: 101
        - Reisbestemming: Kortrijk
        - Aantal zitplaatsen: 5
        - Aantal vrije plaatsen: 2

Detail info over vrachtwagen DAF (2009):
        - ID: 102
        - Reisbestemming: self.reisbestemming
        - Max laadvermogen: 5000
        - Gewicht vracht: 4000.0
*** De vrachtwagen wordt geladen ****
Hoeveel vracht neemt deze vrachtwagen mee? (max: 5000) > 6000
Foutje: Vracht moet kleiner zijn dan het max laadvermogen van de vrachtwagen
```

Werkwijze - inlezen json file

1: Plaast het json-bestand auto.json in de submap ‘doc’.

- Bestudeer aandachtig het json-bestand, controleer waar er [] en {} staan . Je kan zien dat alle Personenwagens van ons taxibedrijf hierin worden bewaard.

2: Voeg in de submap ‘model’ een extra klasse **PersonenwagenRepository** toe om de json in te lezen en te filteren.

3: In **PersonenwagenRepository** programmeer volgende onderdelen:
3a: Voeg een public static methode ‘**inlezen_personenwagens**’ toe die in staat is om het bronbestand in te lezen en een list van personenwagens terug te geven.

3b: Voeg een static methode ‘**zoek_wagen’** toe aan **PersonenwagenRepository** met twee parameters, nl. een list van objecten van de klasse Personenwagen én het id van de wagen dat je wil zoeken.

test_wagens_met_json.py

```
from model.Personenwagen import Personenwagen
from model.PersonenwagenRepository import PersonenwagenRepository


def print_info(lst_auto):
    for auto in lst_auto:
        print(f"{auto.id} - {auto} - heeft {auto.max_aantal_zitplaatsen} plaatsen /
{auto.aantal_personen} passagiers aan boord-> {auto.vrije_plaatsen} vrije plaatsen")

print("****start inlezen****")
beschikbare_taxis = PersonenwagenRepository.inlezen_personenwagens()
print("****inlezen is voltooid****")
input_gebruiker = input("Wat wil je doen? [o]verzicht, [p]assagier ingeven > ")
while input_gebruiker != "":
    if input_gebruiker.upper() == "O":
        print_info(beschikbare_taxis)
    elif input_gebruiker.upper() == "P":
        try:
            temp_id = input("Geef het ID van de taxi > ")
            temp_pers = int(input("Hoeveel personen stappen er in? > "))
```

```
            temp_wagen = PersonenwagenRepository.zoek_wagen(
                beschikbare_taxis, temp_id)
            temp_wagen.aantal_personen = temp_pers
        except Exception as ex:
            print(f"Fout: {ex}")

    input_gebruiker = input(
        "Wat wil je doen? [o]verzicht, [p]assagier ingeven > ")
```

Terminal

```
****start inlezen****
Auto is niet toegevoegd: invalid literal for int() with base 10: 'slechte data'
****inlezen is voltooid****
Wat wil je doen? [o]verzicht, [p]assagier ingeven > o
how100 - Personenwagen MINI 2016 - heeft 4 plaatsen / 0 passagiers aan boord-> 4 vrije
plaatsen
how101 - Personenwagen VW 2019 - heeft 6 plaatsen / 0 passagiers aan boord-> 6 vrije plaatsen
how102 - Personenwagen VW 2017 - heeft 4 plaatsen / 0 passagiers aan boord-> 4 vrije plaatsen
how103 - Personenwagen AUDI 2018 - heeft 5 plaatsen / 0 passagiers aan boord-> 5 vrije
plaatsen
how105 - Personenwagen TESLA 2018 - heeft 5 plaatsen / 0 passagiers aan boord-> 5 vrije
plaatsen
Wat wil je doen? [o]verzicht, [p]assagier ingeven > p
Geef het ID van de taxi > how103
Hoeveel personen stappen er in? > 3
Wat wil je doen? [o]verzicht, [p]assagier ingeven > p
Geef het ID van de taxi > how105
Hoeveel personen stappen er in? > 9
Fout: Er kunnen niet meer personen aanwezig zijn dan dat er zitplaatsen zijn
Wat wil je doen? [o]verzicht, [p]assagier ingeven > o
how100 - Personenwagen MINI 2016 - heeft 4 plaatsen / 0 passagiers aan boord-> 4 vrije
plaatsen
how101 - Personenwagen VW 2019 - heeft 6 plaatsen / 0 passagiers aan boord-> 6 vrije plaatsen
how102 - Personenwagen VW 2017 - heeft 4 plaatsen / 0 passagiers aan boord-> 4 vrije plaatsen
how103 - Personenwagen AUDI 2018 - heeft 5 plaatsen / 3 passagiers aan boord-> 2 vrije
plaatsen
how105 - Personenwagen TESLA 2018 - heeft 5 plaatsen / 0 passagiers aan boord-> 5 vrije
plaatsen
Wat wil je doen? [o]verzicht, [p]assagier ingeven >
```

Studietip:

Als extra oefening kan je een VrachtwagenRepository maken om vrachtwagens.json in te lezen. Maak hiervoor zelf een json file aan.