

Week 9

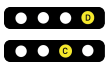
Doelstellingen

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

- Exception handling waar nodig
- File I/O (Json-files) & OOP
- Implementeren van test-methode

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 9 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/ASGJQ9Qe>

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>

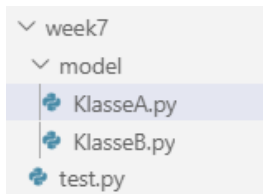
Naming conventions binnen Python: <https://www.python.org/dev/peps/pep-0008/>

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 testklasse!!	<pre>from model.KlasseB import KlasseB # alternatief # from .KlasseB import KlasseB class KlasseA: def __init__(self): self.__var1 = KlasseB()</pre>
Test.py maakt gebruik van de klasseA. We vermelden de submap 'model' waarin hij moet zoeken.	<pre>from model.KlasseA import KlasseA object_a1 = KlasseA() object_a2 = KlasseA()</pre>

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!

Kopieer de startbestanden van leho in week09

Oef 01

Zoek de klasse Bier uit labo week 7 op (of gebruik de klasse Bier uit het bronmateriaal).
Pas deze klasse nu aan zodat

- in elke resp. setter-property een controle op de nieuwe doorgegeven waarde gebeurt
 - *biernaam*, *brouwerijnaam*, *kleur*: nieuwe waarde is een string en mag niet leeg zijn;
 - *alcoholpercentage*: nieuwe waarde is een float en ligt tussen 0 en 100.
- indien de nieuwe waarde niet voldoet, wordt een **ValueError** teruggegeven
- een object van de klasse Bier (later in oef 2) in een list correct getoond wordt.

Test uit door:

- maak een correct bier-object aan. Wijzig nadien de naam van het bier in een lege string.

```
NMCT-Blond (Howest)
Geef een nieuwe biernaam op:>
Foutmelding: Geen geldige biernaam!
```

- maak een bier aan waarbij je voor de brouwerij een lege string doorgeeft.

```
Geef de biernaam op:> nmct++
Geef de brouwerij op:>
Geef de kleur op:> bruin
Geef het alcoholpercentage op:> 12.8
Bier aanmaken...
Foutmelding: Geen geldige brouwerijnaam!
```



Oef 02

Voor de volgende oefeningen kan je je baseren op de besproken demo uit de theorieles.

Download het bronbestand 'bieren.json'. Bestudeer **grondig** het bestand.
Zoals je in de theorieles besproken hebt, zal de opbouw van een json bestand steeds verschillen. De ene keer bestaat het uit een list[], de andere keer uit een dictionary{}.

De doelstelling van deze oefening is om voor elk beschreven bier uit het bestand een object van de klasse Bier aan te maken. Alle gecreëerde objecten houden we in een list bij. Nadien voegen we een extra filterfunctionaliteit toe: het filteren van bieren op basis van een opgegeven brouwerijnaam.

Werkwijze:

- 1: Plaast dit bestand in een afzonderlijke submap 'doc'.
- 2: Voeg een nieuw bestand BierRepository.py in de map 'model' toe. Maak hierin een nieuwe klasse **BierRepository** aan.
- 3: Programmeer volgende onderdelen:
 - Voorzie de klasse van een private klasse-attributte "__filename" die het pad naar het bronbestand bijhoudt. Bijvoorbeeld:

```
__filename = "doc/bieren.json"
```
 - Voeg een public static methode 'inlezen_bieren' toe die in staat is om het bronbestand in te lezen en een list van bieren terug te geven.
 - Om een lokaal json-file in te lezen maakt de public static methode 'inlezen_bieren' gebruik van volgende private static hulpmethode '__inlezen_local_json_file'.
 - importeer bovenaan de json-library zodat je een json file eenvoudig kan bevragen:

```
import json
```

```

BierRepository.py

from .Bier import Bier
import json

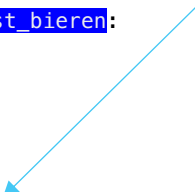
class BierRepository:

    __filename = "doc/bieren.json"

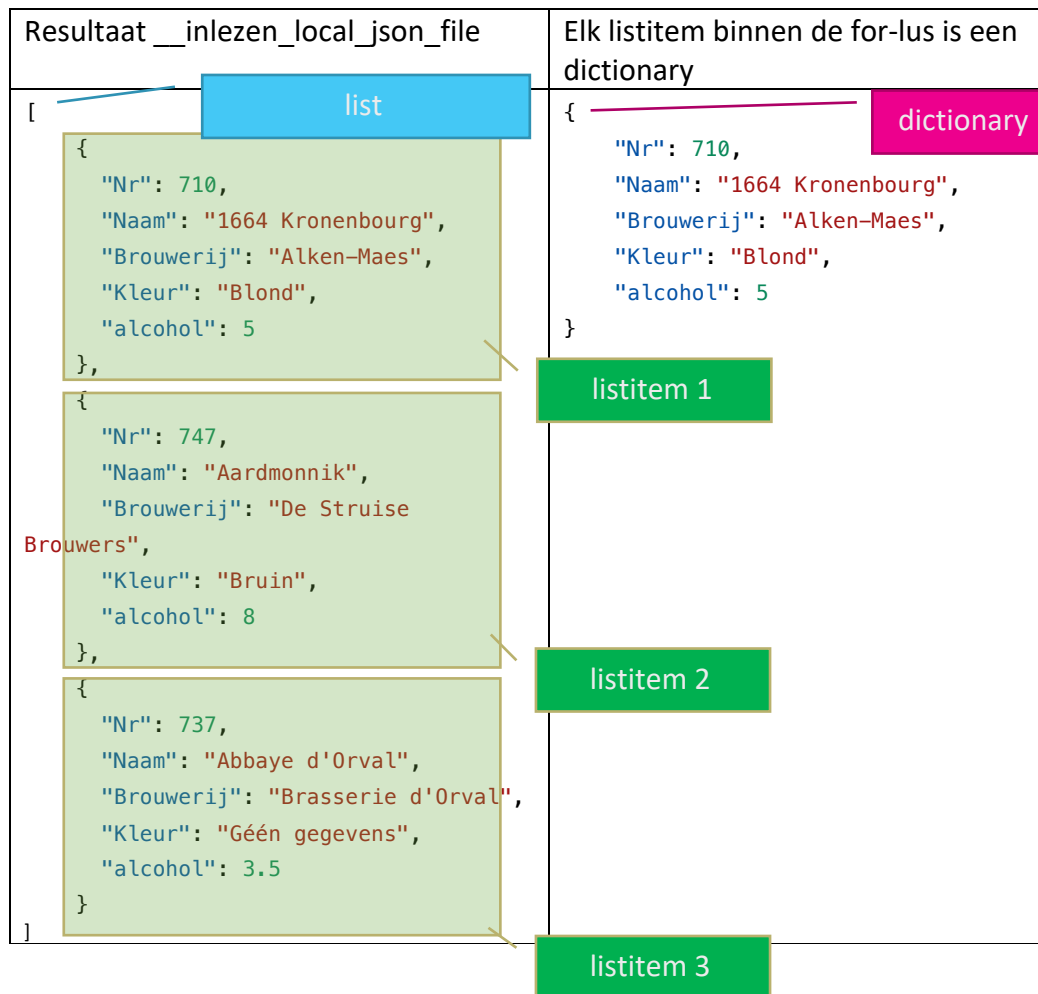
    @staticmethod
    def inlezen_bieren():
        bieren = []
        list_bieren = BierRepository.__inlezen_local_json_file(BierRepository.__filename)
        for dict_bier in list_bieren:
            ...
            ...
        return bieren

    @staticmethod
    def __inlezen_local_json_file(bestandsnaam):
        fo = open(bestandsnaam)
        response_json = fo.read()
        fo.close()
        return json.loads(response_json)

```



De methode `__inlezen_local_json_file` geeft in deze situatie een **list []** terug waarvan elk element op zijn beurt een **dictionary {}** is. Overloop in de methode `inlezen_bieren` binnen de `for`-lus elk element uit deze list. Elk element wordt voorgesteld als een dictionary met bijhorende key en value.



Vraag met de juiste key (**let op voor hoofdletter gebruik in de json file!**) elke waarde op. Maak met alle info een object van de klasse Bier.

- *Opgelet: niet elke vermelde waarde is correct. Gebruik daarom exception handling bij het inlezen.*

Test de methode voldoende uit:

test_oef2_bieren.py
<pre>def test_bieren_a(): # opgave a lijst_bieren = BierRepository.inlezen_bieren() print(f"Aantal correct ingelezen bieren zijn: {len(lijst_bieren)}") print(f"Inhoud van de lijst met bieren: {lijst_bieren}") test_bieren_a()</pre>
<p>Terminal</p> <pre>Foutmelding: Geen geldige bierkleur! Foutmelding: Geen geldige bierkleur! ...(knip) Foutmelding: Geen geldige bierkleur! Foutmelding: Geen geldige bierkleur! Aantal correct ingelezen bieren zijn: 432 Inhoud van de lijst met bieren: [1664 Kronenbourg, Aardmonnik, Aarschotse Bruine, Abbaye d'Orval, Abbaye de Bonne-Esperance, ..., Zottegemse Grand Cru, Zulte oud bruin]</pre>

- Zorg dat de ingelezen list van bier-objecten op basis van het alcoholpercentage van klein naar groot kan gesorteerd worden.
 - Welke operator moet je hiervoor overladen? In welke klasse?

test_oef2_bieren.py
<pre>def print_bieren(lijst_bieren): for bier in lijst_bieren: print(f"{bier} -> alcoholpercentage: {bier.alcoholpercentage}") def test_bieren_b(): # opgave b lijst_bieren = BierRepository.inlezen_bieren() lijst_bieren.sort() print_bieren(lijst_bieren) test_bieren_b()</pre>
Terminal Foutmelding: Geen geldige bierkleur! Foutmelding: Geen geldige bierkleur! ... <i>(knip)</i> Foutmelding: Geen geldige bierkleur! Foutmelding: Geen geldige bierkleur! Star Bruin (Haacht) -> alcoholpercentage: 0.4 Jupiler N.A. (Piedboeuf) -> alcoholpercentage: 0.5 Stella Artois N.A. (Artois) -> alcoholpercentage: 0.5 Tourtel ambrée (Alken-Maes) -> alcoholpercentage: 0.5 Tourtel blonde (Alken-Maes) -> alcoholpercentage: 0.5 Tourtel bruin (Alken-Maes) -> alcoholpercentage: 0.5 Bruno (De Biertoren) -> alcoholpercentage: 0.8 Extra Bruin (Verlinden) -> alcoholpercentage: 0.8 Gulden draak (Bios) -> alcoholpercentage: 12.0 Trappistes Rochefort 10 (Notre Dame de St.-Remy) -> alcoholpercentage: 12.0

- Voeg een static methode 'zoek_bier_uit_brouwerij' toe met parameters een list van objecten van de klasse Bier én een brouwerijnaam. Doorloop alle bieren en geef enkel deze bieren terug afkomstig uit de doorgegeven brouwerij.

test_oef2_bieren.py
<pre>def print_bieren(lijst_bieren): for bier in lijst_bieren: print(f"{bier} -> alcoholpercentage: {bier.alcoholpercentage}") def test_bieren_c(): # opgave c lijst_bieren = BierRepository.inlezen_bieren() lijst_bieren.sort() deel_brouwerij = input("\nGeef (een deel van) de brouwerijnaam op: ") print("Gevonden bieren zijn: ") resultaat = BierRepository.zoek_bieren_uit_brouwerij(lijst_bieren, deel_brouwerij) print_bieren(resultaat) test_bieren_c()</pre>
Terminal Foutmelding: Geen geldige bierkleur! Foutmelding: Geen geldige bierkleur! ... <i>(knip)</i> Foutmelding: Geen geldige bierkleur! Foutmelding: Geen geldige bierkleur! Geef (een deel van) de brouwerijnaam op: bavik

Gevonden bieren zijn:

Petrus (Bavik) -> alcoholpercentage: 4.5

Pony-Stout (Bavik) -> alcoholpercentage: 4.5

Speciale Stop (Bavik) -> alcoholpercentage: 4.5

Super Pils (Bavik) -> alcoholpercentage: 4.5

Cuvée St. Amand (Bavik) -> alcoholpercentage: 7.5

Oef 03

Voor deze oefening maken we gebruik van een json-file afkomstig van een webservice (<https://makeup-api.herokuapp.com/>) Deze webservice laat de gebruiker toe informatie over talrijke make-up producten op te vragen. Afhankelijk van de interesse kan men de zoekopdracht verfijnen tot een specifiek producttype en/of subcategorie. Hoe verfijnder de zoekopdracht, hoe sneller het antwoord binnenkomt. Bijvoorbeeld, volgende link geeft enkel info over lipsticks terug.

https://makeup-api.herokuapp.com/api/v1/products.json?product_category=lipstick&product_type=lipstick

In het bronmateriaal vind je 'makeupproducts.json' met het resultaat van bovenstaande zoekopdracht terug. *(het staat je vrij om de inhoud te vervangen door het resultaat van een andere zoekopdracht).*

Bestudeer aandachtig het json-bestand, en vervul volledig volgende uitspraken.

- Het resultaat van de json is een[] van{}
- Elke{} bestaat uit keys en values én stellen een makeup artikel voor.
 - De key *product_colors* is speciaal, deze bestaat uit een [] van{}

```
{
  "id": 1046,
  "brand": "colourpop",
  "name": "Lippie Stix",
  "price": "5.5",
  "price_sign": "$",
  "currency": "CAD",
  "image_link": "https://cdn.shopify.com/s/files/1/000/000/000/products/colourpop-lippie-stix-formula.jpg",
  "product_link": "https://colourpop.com/products/colourpop-lippie-stix-formula",
  "website_link": "https://colourpop.com/products/colourpop-lippie-stix-formula",
  "description": "Lippie Stix Formula",
  "rating": null,
  "category": "lipstick",
  "product_type": "lipstick",
  "tag_list": [
  ],
  "created_at": "2018-07-08T21:47:49.4",
  "updated_at": "2018-07-09T00:53:23.2",
  "product_api_url": "https://makeup-api.herokuapp.com/api/v1/products.json?id=1046",
  "api_featured_image": "https://s3.amazonaws.com/makeup-api-heroku-production/images/colourpop-lippie-stix-formula.jpg",
  "product_colors": [
    {
      "hex_value": "#5a1e1b",
      "colour_name": "Cola Pop"
    },
    {
      "hex_value": "#641126",
      "colour_name": "Berry Pop"
    }
  ]
}
```

Voor deze oefening wensen we van elk product volgende informatie bij te houden:

- id
- brand
- name
- price
- product_link
- product_colors

Merk op dat voor de elk beschikbaar productkleur volgende info wordt getoond: hex_value en een color_name.

Doel: inlezen van het json-bestand waarbij een lijst van producten wordt aangemaakt. In elk product zit een lijst met beschikbare productkleuren.

Werkwijze:

1: Plaast het json-bestand in de submap 'doc'.

Verken het json bestand eens via verkenning_json.py. Dit bestand heb je straks niet meer nodig, maar dient enkel om je de keys en values eigen te maken.

2: In de submap 'model' maken we volgende twee dataklassen aan:

2a: klasse '**ProductColor**' met de attributen *colorname* en *hexvalue*. Voorzie de klasse van een constructor, properties, tostring-methode, repr-methode, eq-methode

2b: klasse '**MakeUpProduct**' met de attributen *id*, *brand*, *name*, *price*, *productlink* en *colors* aan.

- De klassieke methodes `__init__()` met parameters *id*, *brand*, *name*, *price*, *productlink*
- Voor de colors maak je een lege list aan in de init-methode. Je voorziet voor dit attribuut enkel een get-property. Voor de andere attributen voorzie je een get&set property.
- Voeg een methode '**add_productcolor**' toe: deze methode heeft één parameter, nl een object van de klasse ProductColor. Deze methode voegt het object aan de lijst met colors toe.
 - o Controleer in de methode of de parameter weldegelijk een object van de klasse ProductColor is.
 - o Controleer of de parameter niet in de list al aanwezig is.
(welke methode uit de klasse ProductColor wordt hiervoor achter de schermen gebruikt?)
- Programmeer ook de methode `__str__()`: deze geeft terug:
name (brand) -> Available Colors: aantal_beschikbare_kleuren

3: Voeg een nieuw bestand MakeUpRepository.py in de map 'model' toe. Maak hierin een nieuwe klasse **MakeUpRepository** aan.

- Voorzie de klasse van een private klasse-attributte “__filename” die het pad naar het bronbestand bijhoudt. Bijvoorbeeld:

```
__filename = "doc/makeupproducts.json"
```

- Voeg een static methode ‘load_products’ toe die in staat is om het bronbestand in te lezen en een list van producten terug te geven.

Om een lokaal json-file in te lezen maak je gebruik van volgende private hulpfunctie:

- Deze hulpmethode geeft een list terug, met dictionaries. Elk element (dictionary) bevat informatie voor één MakeUpProduct. Overloop nu alle elementen. Elk element is een dictionary waarmee je specifieke informatie kan opvragen.

Vraag via de juiste key (let op hoofdletters) elke waarde op. Maak met alle info een object van de klasse MakeUpProduct. *Opgelet: niet elke vermelde waarde is correct. Gebruik daarom exception handling.*

```
class MakeUpRepository:

    __filename = "doc/makeupproducts.json"

    @staticmethod
    def load_products():
        pass

    @staticmethod
    def __inlezen_local_json_file(bestandsnaam):
        fo = open(bestandsnaam)
        response_json = fo.read()
        fo.close()
        return json.loads(response_json)
```

Test de methode voldoende uit in een test-bestand ‘test_makeup.py’, vergeet niet de correcte klassen te importeren

test_makeup.py
...
<pre>list_products = MakeUpRepository.load_products() print(f"Aantal ingelezen make-up producten: {len(list_products)}")</pre>
Terminal Aantal ingelezen make-up producten: 122

4: Voeg een static methode 'search_in_products' toe aan MakeUpRepository met twee parameters, nl. een list van objecten van de klasse MakeUpProduct én een deel van een productnaam. Doorloop alle objecten uit de list en geef enkel deze terug waarvan de productnaam het gezochte deel bevat.

Voordat je de producten toont, **sorteer** je de volledig lijst van MakeUpProducten. Producten met het minst aantal kleuren staan vooraan in de lijst.

Vervolledig 'test_makeup.py', om volgende output.

test_makeup.py
... list_products = MakeUpRepository.load_products() print(f"Aantal ingelezen make-up producten: {len(list_products)}") ...
Terminal Aantal ingelezen make-up producten: 122 Geef een deel van de naam op:> dior Dit zijn de gevonden producten: Dior Holiday Couture Collection (dior) -> Available Colors: 0 ROUGE DIOR COLLECTION COUTURE - Christmas Look 2017 Limited Edition (dior) -> Available Colors: 1 DIORIFIC MATTE FLUID - Christmas Look 2017 Limited Edition (dior) -> Available Colors: 2 ROUGE DIOR - Fall 2017 Limited Edition (dior) -> Available Colors: 4 DIORIFIC KHÔL (dior) -> Available Colors: 5 Dior Addict Lipstick - Limited Edition (dior) -> Available Colors: 5 Rouge Dior Double Rouge (dior) -> Available Colors: 6 Diorific (dior) -> Available Colors: 10 DIOR ADDICT LACQUER STICK (dior) -> Available Colors: 20 Rouge Dior Liquid (dior) -> Available Colors: 24 Dior Addict Lipstick (dior) -> Available Colors: 35 Rouge Dior (dior) -> Available Colors: 36