

Week 7

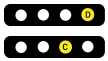
Doelstellingen

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

- Aanmaak van data-klassen met hierbij
 - Integratie van de juiste instantie-attributen met de juiste accessscope
 - Integratie van property-/setter-methodes
 - Integratie van methodes `__init__()`, `__del__()`, `__str__()`, `__repr__()`,...
- Test-methodes

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 7 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/agmkqSKk>

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 zoals in het labo. Het is pas door de oefeningen thuis “alleen” te maken dat je de leerstof zich 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 6

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/>

Oefeningen

Vermeld in commentaar telkens de opgave!


Installeer volgende plugin: deze plugin vind je terug op Leho. Zie klassikale demo voor verdere toelichting.



Snippet for Basic Programming Skills v0.0.7

Howest-MCT

Snippet for Basic Programming Skills MCT/MIT

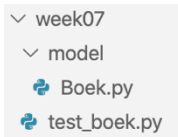
Disable Uninstall  This extension is enabled globally.



Oef 01

Maak een dataklasse **Boek**. Wat zijn de kenmerken waarmee een boek zich kan identificeren? Codeer nu deze dataklasse:

- Maak in je folder een subfolder “model” aan, in deze folder creeër je de file Boek.py
- In de rootfolder maak je een file test_boek.py



- Voorzie de klasse Boek van de nodige private attributen
 - *titel, auteur, uitgeverij, isbn, jaargang*
- Maak voor elk attribuut een publiek **getter** en **setter** property-methode aan.
- Programmeer de constructor: de methode `__init__()`.
 - Deze heeft 5 parameters: *titel, auteur, uitgeverij, isbn, jaargang*
 - Zoals je kan afleiden uit de testcode, is de vijfde parameter optioneel. Wordt de *jaargang* niet opgegeven in de constructor dan krijgt het de default waarde 2020.
- Programmeer de methode `__str__()`
 - De gewenste output van deze methode is “*auteur, titel (uitgeverij) *jaargang**”

Test uit door verschillende objecten van deze klasse aan te maken. Gebruik hiervoor een afzonderlijk bestand test_boek.py. Je kan hiervoor ook het testbestand op Leho gebruiken.

```
# Test class test_boek.py

from model.Boek import Boek

b1 = Boek("Python for dummies", "Louis Kegels", "Arco", "125-875-5455")
print(f"Titel van boek 1: {b1.titel}")
b1.titel = "Python for dummies bis"
print("Volledige info boek 1")
print(b1)

b2 = Boek("Hoe leg ik een vijver in mijn tuin aan? Deel 1",
        "Johan Vannieuwenhuyse", "Aveve", "987-875-5455", 2016)
print("Volledige info boek 2")
print(b2)
```

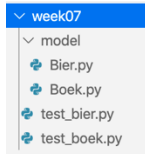
```
Terminal:
Titel van boek 1: Python for dummies
Volledige info boek 1
Louis Kegels , Python for dummies bis (Arco) *2019*
Volledige info boek 2
Johan Vannieuwenhuyse , Hoe leg ik een vijver in mijn tuin aan? Deel 1 (Aveve) *2016*
```



Oef 02

Bestudeer het bronbestand `bieren.csv`.

Door welke zaken wordt een bier gekenmerkt? Maak nu een dataklasse **Bier**.



- Voorzie de klasse van de nodige (private) attributen: *naam*, *brouwerij*, *alcoholpercentage*, *kleur*
- Maak voor elk attribuut een (publieke) property aan.
 - In de **setter** property voor *naam*, *brouwerij* en *kleur*
 - Wordt de nieuwe waarde gecontroleerd: een lege string wordt als 'onbekend' bewaard.
 - In de **setter** property voor *alcoholpercentage* controleer je of
 - Het een float is.
 - Het tussen 0 en 100 ligt.
 - Zoniet bewaar je de waarde -1 in zijn private attribuut.
 - De **getter** property voor *naam* wordt altijd in hoofdletters weergegeven.
- Programmeer de constructor: de methode `__init__()`
 - De parameters zijn de vier attributen.
- Programmeer de methode `__str__()`
 - De output is "*naam brouwerij - alcoholpercentage*"

Maak verschillende bieren aan. Wijzig nadien de attributen via de setter-property. Controleer of de functionaliteit in orde is. Je kan hiervoor ook het testbestand op Leho gebruiken.

```
# Test class test_bier.py
from model.Bier import Bier

print("*** Bier 1 ***")
b1 = Bier("Augustijn", "Bios", 12.0, "amber")
print(b1)
# verander de brouwerij naar lege string
b1.brouwerij = ""
print(b1)
```

Terminal

```
** Bier 1 ***
AUGUSTIJN Bios - 12.0
AUGUSTIJN onbekend - 12.0
```

```
# Test class test_bier.py
from model.Bier import Bier

print("** Bier 2 **")
b2 = Bier("Jupiler", "", 3.3, "blond")
print(f"Het kleur van {b2.naam} is {b2.kleur} ")
print(f"Alcoholpercentage: {b2.alcoholpercentage} ")
b2.alcoholpercentage = "5,5"
print(f"Alcoholpercentage: {b2.alcoholpercentage} ")
```

Terminal

```
** Bier 2 **
Het kleur van JUPILER is blond
Alcoholpercentage: 3.3
Alcoholpercentage: -1
```



Oef 03

Maak een dataklasse **Hotelgast** en testklasse **test_hotelgast**. Van een hotelgast bewaar je in de (private) attributen

- de *naam*
- de *voornaam*
- het openstaand *saldo*
- of de gast al dan niet is *ingecheckt* (True/False)

In de publiek **setter** property doe je volgende controle voordat je de waarde bewaart in de overeenkomstige attributen:

- de *naam* mag niet leeg zijn, zoniet bewaar je ONBEKEND
- de *voornaam* mag niet leeg zijn, zoniet bewaar je ONBEKEND
- Het *saldo* moet een positief geheel getal zijn. Geeft de gebruiker een negatieve waarde in, dan wordt het saldo toch als 0 (euro) bewaard.
- Je controleert of het type van *is_ingecheckt* een bool(ean) is.
 - Is het een ander gegevenstype, dan bewaar je de waarde False.

Programmeer de constructor of de methode `__init__()`

- De parameters zijn *naam*, *voornaam*, *saldo* en *is_ingecheckt*.

Programmeer de methode `__str__()`

- De output voor ingecheckte personen is “OK: FAMILIENAAM – saldo euro”
- De output voor uitgecheckte personen is “X: Voornaam FAMILIENAAM”

Controleer of de functionaliteit in orde is. Je kan hiervoor ook het testbestand op Leho gebruiken.

Merk op dat er in de getter property van familienaam niet gevraagd wordt om naar hoofdletter te worden omgezet.

- Dus `print(gast1.familienaam)` zal gewoon “Walcarius” geven.
- Waar `print(gast1)` “OK: WALCARIUS - 0 euro” geeft.

```
from model.Hotelgast import Hotelgast

gast1 = Hotelgast("Walcarius", "Stijn", -100, True)
gast2 = Hotelgast("Roobrouck", "Dieter", 30, False)

print("*** Volgende gasten verblijven bij ons: ")
print(gast1)
print(gast2)
print("**** checkt uit met fout")
gast1.is_ingecheckt = "gaat weg"
print(gast1)
```

Terminal

```
*** Volgende gasten verblijven bij ons:
OK: WALCARIUS - 0 euro
X: Dieter ROOBROUCK
**** checkt uit met fout
X: Stijn WALCARIUS
```



Oef 04

Maak een dataklasse Auto waarbij volgende zaken worden bijgehouden: *merk*, *model*, *kleur*, *brandstof*, *km-stand*.

- Maak de gewenste **setter** en **getter** properties.
 - Hoe kan je er voor zorgen dat enkel de attributen *kmstand* en *kleur* vanuit de testklasse gewijzigd kunnen worden?
- Programmeer de constructor: de methode `__init__()`
 - De parameters zijn volgende attributen
 - *merk*, *model*, *kleur*, *brandstof*
 - *kleur* en *brandstof* heeft een default waarde van “grijs” en “diesel”.
 - Binnen de constructor geef je de *kilometerstand* een waarde 0. (Deze komt **niet** binnen via een parameter)
- Programmeer de methode `__str__()`
 - De output is “*merk* (model: *model* kleur: *kleur*)”
- Voeg een extra methode ‘rijden’ met de parameter *extra_km* toe: hiermee wordt de km-stand van de auto verhoogd.
- Voeg een methode “maak_lawaai” toe die
 - de string ***bwaahrooah*** returnt voor een ***diesel*** auto
 - de string ***swooaahsj*** returnt voor een ***benzine*** auto
 - de string ***sssssssst*** returnt voor een ***elektrisch*** auto
 - de string ***panne*** returnt voor alle andere situaties

Test alles uit: maak een list aan met verschillende voertuigen. Laat vervolgens elk voertuig uit de list een random afstand afleggen. Print nadien van elk voertuig de km-stand af.

Je kan hiervoor ook het testbestand op Leho gebruiken.

```
import random
from model.Auto import Auto

autos = []
autos.append(Auto("Volkswagen", "passat", "donkergrijs", "diesel"))
autos.append(Auto("Opel", "astra", "groen", "benzine"))
autos.append(Auto("Seat", "ibiza", "blauw", "diesel"))

for auto in autos:
    print(f"Auto {auto} heeft op de kmteller {auto.kmstand}")
print("\nOp het einde van de dag: ")
for auto in autos:
    auto.rijden(random.randint(10, 300))
    print(f"\tAuto {auto} heeft op de kmteller {auto.kmstand} en doet {auto.maak_lawaai()}")
```

Terminal

```
Auto Volkswagen (model: passat kleur: donkergrijs) heeft op de kmteller 0
Auto Opel (model: astra kleur: groen) heeft op de kmteller 0
Auto Seat (model: ibiza kleur: blauw) heeft op de kmteller 0

Op het einde van de dag:
Auto Volkswagen (model: passat kleur: donkergrijs) heeft op de kmteller 118 en doet bwaahrooah
Auto Opel (model: astra kleur: groen) heeft op de kmteller 257 en doet swooaahsj
Auto Seat (model: ibiza kleur: blauw) heeft op de kmteller 271 en doet bwaahrooah
```




Oef 05

We wensen van elk wiel de straal en het aantal omwentelingen bij te houden. Hiervoor maken we een klasse **Meetwiel** met de attributen *straal* en *omwentelingen*.

Volgend stappenplan kan gevolgd worden:



- Voorzie de klasse van de nodige (private) attributen
- Voorzie de bijhorende **getter** en **setter** property-methodes.
- Voorzie ook **2 extra getter** property-methodes:
 - Deze twee properties hebben geen bijhorende attributen (opslagplaats), dit komt omdat ze gebruik maken van de andere attributen om "iets" te berekenen. Hierdoor hebben ze ook geen setter property.
 - Programmeer de **getter property**-methode *omtrek*: deze geeft de omtrek van het wiel terug
 - Programmeer de **getter property**-methode *afstand*: deze geeft de afgelegde afstand ifv de het aantal *omwentelingen* en de omtrek van het *meetwiel* terug
- Programmeer de constructor: methode `__init__()`
 - Heeft de volgende optionele parameters *straal* en *omwentelingen* (default-waarde voor deze 2 parameters is 0).
- Programmeer de methode `__str__()`
 - Geeft volgende output:
Meetwiel met straal *straal* en omwentelingen *omwentelingen*.

Test voldoende uit door verschillende objecten van deze klasse aan te maken. Je kan hiervoor ook het testbestand op Leho gebruiken.

Vraag tenslotte aan de gebruiker meerdere extra omwentelingen voor een wiel op. Sluit af met 'c'. Print nadien de afgelegde afstand opnieuw af. Voorbeeld:

```
from model.Meetwiel import Meetwiel

meetwiel1 = Meetwiel(0.9) # straal
meetwiel2 = Meetwiel(0.45, 123) # straal , omwentelingen
print(meetwiel1)
print(meetwiel2)

waarde = input(
    "Geef het aantal extra omwentelingen door of sluit af met 'c':> ")
while (waarde != "c" and waarde.isnumeric()):
    meetwiel1.omwentelingen += int(waarde)
    waarde = input(
        "Geef het aantal extra omwentelingen door of sluit af met 'c':> ")
print(meetwiel1)
print(f"Meetwiel 1 legde reeds {meetwiel1.afgelegde_afstand:.2f} m af")
```

Terminal

```
Meetwiel met straal 0.9 en omwentelingen 0
Meetwiel met straal 0.45 en omwentelingen 123
Geef het aantal extra omwentelingen door of sluit af met 'c':> 2
Geef het aantal extra omwentelingen door of sluit af met 'c':> 1
```

```
Geef het aantal extra omwentelingen door of sluit af met 'c':> 3  
Geef het aantal extra omwentelingen door of sluit af met 'c':> c  
Meetwiel met straal 0.9 en omwentelingen 6  
Meetwiel 1 legde reeds 33.93 m af
```



Oef 06

Oef: maak een dataklasse **Winkelkar** die als private attribuut een list van producten (String) bijhoudt. Voorzie de klasse van volgende methodes:

- Programmeer de constructor: methode `__init__()`
 - Deze methode maakt altijd een lege winkelkar aan en heeft dus geen parameters.
De producten worden voorgesteld als een lege list in een private attribuut.
- Programmeer de methode `__str__()`
- Een **getter** property-methode 'producten' die de lijst terug geeft
- **Methode** 'voeg_product_toe(nieuw_product)' die aan het winkelkarretje een nieuw product toevoegt.
- **Methode** 'verwijder_product(product)' die uit het winkelkarretje een product verwijdert.
 - Wat gebeurt er als je een product wil verwijderen die niet bestaat? Hoe kan je dit vermijden?

Test volgende code uit. Je kan hiervoor het testbestand op Leho gebruiken.

- Maak 2 winkelkarretjes aan. Voeg verschillende producten aan elk toe. Print nadien beide af.

```
from model.Winkelkar import Winkelkar
```

```
action_kar = Winkelkar()
action_kar.voeg_product_toe("cd1")
action_kar.voeg_product_toe("cd2")
action_kar.voeg_product_toe("cd3")
action_kar.voeg_product_toe("cd4")
action_kar.verwijder_product("cd3")
ikea_kar = Winkelkar()
ikea_kar.voeg_product_toe("Billy")
ikea_kar.voeg_product_toe("Factum")
```

```
print(f"Winkelkar 1: {action_kar}")
print(f"Winkelkar 2: {ikea_kar}")
```

Terminal

```
Winkelkar 1: De winkelkar bestaat uit 3: ['cd1', 'cd2', 'cd4']
Winkelkar 2: De winkelkar bestaat uit 2: ['Billy', 'Factum']
```

Oefeningen voor thuis



Thuis 1

Maak een dataklasse **Postpakket**. Van een postpakket bewaar je volgende items: *omschrijving*, *breedte*, *hoogte* en *diepte*.

- Voorzie de klasse van de nodige (private) attributen en bijhorende getter en setter property-methodes.
- In de (publiek) setter property-methodes doe je volgende controle voordat je de waarde bewaart in de overeenkomstige attributen:
 - De breedte, hoogte en diepte moet een geheel getal zijn.
 - De breedte, hoogte en diepte moet groter zijn dan 0 (cm). Anders geef je het de waarde 1 (cm)
- De *omschrijving* mag niet gewijzigd worden. Deze heeft enkel een (publieke) **getter** property-methode.
 - Tip: welk effect zal dit straks hebben binnen in de constructor `__init__()`?
- Er is nog één extra **getter** property-methode die **geen** attribuut heeft.
 - Deze geeft het volume terug ($\text{breedte} * \text{hoogte} * \text{diepte}$)
- Programmeer de methode `__init__()`
 - Heeft de volgende parameters *omschrijving*, *breedte*, *hoogte* en *diepte*
- Programmeer de methode `__str__()`
 - Geeft volgende output:
 - Pakketje: *omschrijving* (*breedte* cm * *hoogte* cm * *diepte* cm)

Test volgende code uit. Je kan hiervoor het testbestand op Leho gebruiken.

```
from model.Postpakket import Postpakket

bol = Postpakket("GSM", 3, 3, 4)
print(bol)

print(f"Volume van pakket is: {bol.volume:.2f}")

amazon = Postpakket("Alexa", 3.3, 34, 3)
print(amazon)
```

Terminal

```
Pakketje: GSM (3 cm * 3 cm * 4 cm)
Volume van pakket is: 36.00
Pakketje: Alexa (1 cm * 34 cm * 3 cm)
```



Thuis 2

Maak een dataklasse **Breuk** aan waarbij we de *teller* en *noemer* gaan bijhouden. Volg opnieuw volgend stappenplan.

Basis:

- Programmeer de methode `__init__()`
 - De parameters zijn *teller* en *noemer*
- Voorzie voor elk vermeld attribuut een property-methode.
- Programmeer de methode `__str__()`

Uitbreiding:

- Voorzie een public methode `vereenvoudig()`: hierbij wordt gekeken of teller en noemer door berekening van de grootste gemene deler kunnen vereenvoudigd worden
 - Python voorziet een functie om de grootst gemene deler te bepalen.
<https://docs.python.org/3.8/library/math.html#math.gcd>
- (na de theorie van week 8) Programmeer de methode `__add__(...)` die toelaat om twee breuken te laten optellen via de `+`-operator. Werk zelf de andere methodes uit: verschil, product en deling.

Test voldoende uit door verschillende objecten van deze klasse aan te maken. Test vervolgens de methode 'vereenvoudig', de bewerkingen tussen breuken onderling.

Je kan hiervoor het testbestand op Leho gebruiken.

```
Breuk 1:
3 / 4
Noemer wijzigen naar 6:
3 / 6
Breuk 1 laten vereenvoudigen:
1 / 2

Breuk 2 na vereenvoudiging:
2 / 5
Som van Breuk 1 en Breuk 2:
9 / 10
```