

## Week 8

### 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__()`,...
  - Gebruik van static variabelen en methodes
- Je kan overweg met associaties tussen klassen.
- 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 8 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/yO\\_ndmzq](https://classroom.github.com/a/yO_ndmzq)

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 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 monitorea!

## Feedback labo week 7

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

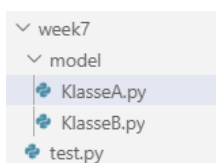
Officiële documentatie van Python: <https://docs.python.org/3.7/>  
Handige tutorial: <https://docs.python.org/3.7/tutorial/index.html>  
Naming conventions binnen Python: <https://www.python.org/dev/peps/pep-0008/>

## 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. Het <b>punt is noodzakelijk</b> en doet Klasse A, zoeken naar Klasse B in <b>dezelfde</b> map als zichzelf!	<pre>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>

Samengevat: je kijkt steeds vanuit het standpunt van de klasse waarin je iets moet importeren.

## Oefeningen

Vermeld in commentaar telkens de opgave!



### Afwerking labo week 7:

Werk de opgaven van laboweek 7 af. In de startbestanden vind je de laatste versie van de klasse Winkelkar terug.

- Zorg er voor dat de +-operator toegepast kan worden:
  - zodat twee winkelkarretjes bij elkaar kunnen opgeteld worden en een nieuw winkelkarretje opleveren (welke methode moet hiervoor toegevoegd worden?)
  - zodat het bestaande winkelkarretje uitgebreid wordt met de producten uit een andere winkelkarretje (welke methode moet hiervoor toegevoegd worden?)
- Tel beide winkelkarretjes op via de plus-operator. Print resultaat 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}")
print("***** Winkelkar 3 = Winkelkar 1 + Winkelkar 2 *****")
kerst_kar = action_kar + ikea_kar
print(f"Winkelkar 3: {kerst_kar}")
print("***** Winkelkar 1 +=Winkelkar 2 *****")
action_kar += ikea_kar
print(f"Winkelkar 1: {action_kar}")
```

#### Terminal

```
Winkelkar 1: De winkelkar bestaat uit 3: ['cd1', 'cd2', 'cd4']
Winkelkar 2: De winkelkar bestaat uit 2: ['Billy', 'Factum']
***** Winkelkar 3 = Winkelkar 1 + Winkelkar 2 *****
Winkelkar 3: De winkelkar bestaat uit 5: ['cd1', 'cd2', 'cd4', 'Billy',
'Factum']
***** Winkelkar 1 +=Winkelkar 2 *****
Winkelkar 1: De winkelkar bestaat uit 5: ['cd1', 'cd2', 'cd4', 'Billy',
'Factum']
```



## Oef 01

Je vertrekt van het startbestand Speler.py. Overloop de code: ga na welke properties/data-attributen aanwezig zijn.

Breid de klasse uit:

- de methode "maak\_doelpunt" verhoogt het aantal eigen doelpunten van een speler
- de methode "?????????????" zodat een speler correct wordt afgeprint binnen een list.
- de ploegnaam wordt gedeeld over de volledige ploeg (public static variabele)
- de doelpunten van het volledige team: dit is de som van alle individuele doelpunten. Dus iedere maal een speler een doelpunt maakt wordt het doelpuntsaldo van de ploeg ook verhoogd. (private static)  
Vergeet dit ook niet te doen tijdens de \_\_init()\_\_
- een public static-methode om deze score terug te geven



Aantal eigen doelpunten speler



Waarde van de speler (score op 10)

Ploeg = rode duivels

Aantal doelpunten ploeg = 17

### Test klasse

```
from model.Speler import Speler
from model.Geboortedatum import Geboortedatum

# aanspreken van een public static variabele (attribute)
Speler.naam_ploeg = "Rode duivels"

sp1 = Speler("Thibault", "Cortois", "keeper", 8, 0)
sp2 = Speler("Vincent", "Kompany", "aanvaller", 8, 3)
# par 4 en 5 worden niet opgegeven, zie default par in classe __init__()
sp3 = Speler("Axel", "Witsel", "aanvaller")

team = [sp1, sp2, sp3]
print(team)

print("\nVincent scoort!")
sp2.maak_doelpunt()
print(sp2)

print("\nAxel scoort!")
sp3.maak_doelpunt()
print(sp3)

print(f"Het doelpunten saldo van { Speler.naam_ploeg } is
{Speler.get_doelpunten_saldo_ploeg()}")
```

#### Terminal

```
[speler: Cortois, Thibault (8/10), speler: Kompany, Vincent (8/10), speler: Witsel,
Axel (0/10)]
```

```
Vincent scoort!
speler: Kompany, Vincent (8/10) doelpunten: 4
```

```
Axel scoort!
speler: Witsel, Axel (0/10) doelpunten: 1
Het doelpunten saldo van Rode duivels is 5
```



## Oef 02

Maak een dataklasse **Geboortedatum**. Zorg ervoor dat *dag*, *maand* en *jaar* bijgehouden worden.

Voorzie nu zelf de klasse van:

- Voorzie de klasse van de nodige (eventueel) private attributen: *dag*, *maand*, *jaar*
- Maak voor elk attribuut een (eventueel) publieke property aan.  
Bouw controles in wanneer dag/maand/jaar gewijzigd worden.
  - Tussen welke waarde moet een dag/maand liggen?
- Programmeer de methode `__init__()`
  - Met 3 parameters *dag*, *maand*, *jaar*
- Programmeer de methode `__str__()`
  - Met volgende output "*dag/maand/jaar*"

Test de klasse uit!

Breid de klasse vervolgens uit zodat het aantal gecreëerde objecten van de klasse **Geboortedatum** bijgehouden wordt.

- Maak hiervoor bovenaan een private static variabele `__aantal_geboortedata` aan.
- Maak een public static methode '**geef\_aantal\_geboortedatums**' om deze property op te vragen.

Test de klasse uit!

Breid de klasse uit met methodes die één random geboortedatum teruggeeft.

- Maak een public static methode '**genereer\_willekeurige\_verjaardag**'
- Maak een public static methode '**genereer\_lijst\_verjaardagen**' die een list van willekeurige geboortedatums teruggeeft. De parameter is het gewenste aantal.

Test elke nieuwe methode uit!

Tot slot:

- Integreer de methode `__eq__()` die instaat om de gelijkheid tussen twee geboortedatums te controleren. Bepaal eerst vooraf wanneer twee geboortedatums aan elkaar gelijk zijn.
- Wat merk je op als je een list met Geboortedata afprint? De `__str__()` methode werkt niet in deze situatie? Welke methode moet je hiervoor voorzien?
- Maak een (gewone) methode **zelfde\_verjaardag()** met als parameter een object **Geboortedatum**, die controleert of beide op dezelfde dag verjaren (m.a.w. dag & maand zijn gelijk). Welk datatype kies als return waarde?
- Test uit door verschillende objecten van deze klasse aan te maken en te vergelijken met elkaar.

### Test klasse

```
from model.Geboortedatum import Geboortedatum

# controle
datum1 = Geboortedatum(25, 9, 1977)
print(datum1)
datum1.dag = 32 # controle testen
print(datum1)

datum2 = Geboortedatum(25, 9, 1977)
if (datum1 == datum2):
    print("gelijk")
else:
    print("niet gelijk")

print("Willekeurige lijst geboortedatums: ")
aantal = int(input("Hoeveel geboortedatums wenst u? "))
geboortedatums = Geboortedatum.genereer_lijt_geboortedata(aantal)
index = 1
for datum in geboortedatums:
    print(f"{index} : {datum} ")
    index += 1

print(
    f"Totaal gecreëerde objecten van de klasse Geboortedatum:
    {Geboortedatum.geef_aantal_geboortedatums()}")
```

#### Terminal

```
25/9/1977
-1/9/1977
niet gelijk
Willekeurige lijst geboortedatums:
Hoeveel geboortedatums wenst u? 3
1 : 19/2/2012
2 : 5/2/1953
3 : 18/5/1978
Totaal gecreëerde objecten van de klasse Geboortedatum: 5
```





## Oef 03

We integreren vorige oefeningen in elkaar. Zorg ervoor dat van elke speler ook de *geboortedatum* (object van de klasse *Geboortedatum*) wordt bijgehouden.

- Maak een property *geboortedatum* (en bijhorend attribuut) in de klasse *Speler* aan.
- Breid de constructor uit met een extra (optionele) parameter. Geef deze parameter een default-value, nl. 1 jan 2019

Print van elke aangemaakte speler nu ook zijn geboortedatum af.

Test klasse
<pre>from models.Speler import Speler  def test_spelers_oef3():     sp1 = Speler("Thibault", "Cortous", "keeper", 8, 0, Geboortedatum(11,5,1992))     sp2 = Speler("Vincent", "Kompany", "aanvaller", 8, 3, Geboortedatum(10,4,1986))     sp3 = Speler("Axel", "Witsel", "aanvaller")      print("De geboortedata van de spelers zijn:")     for item in [sp1, sp2, sp3]:         print(f"{item} -&gt; geboortedatum: {item.geboortedatum}")  test_spelers_oef3()</pre>
<b>Terminal</b> De geboortedata van de spelers zijn: speler: Cortous, Thibault (8/10) doelpunten: 0 -> geboortedatum: 11/5/1992 speler: Kompany, Vincent (8/10) doelpunten: 3 -> geboortedatum: 10/4/1986 speler: Witsel, Axel (0/10) doelpunten: 0 -> geboortedatum: 1/1/2019



## Oef 04a

Voor deze opgave staat de klasse `Hotelgast` uit het labo van week 7 reeds gekopieerd in het startbestand.

Maak nu een nieuwe klasse `Hotel` die naast de **naam** van het hotel ook de **gastenlijst** bijhoudt.

Voorzie de klasse `Hotel` nu van

- De klassieke methodes `__init__()` en `__str__()`
  - Je kan de parameters van de constructor afleiden uit de test klasse.
- **Property** om de naam van het Hotel **op te vragen** en te **wijzigen** (controleer of de naam geen lege string is)
- Voor de gasten maak je een lege list aan in de init-methode. Daarna voorzie je hiervoor enkel een get-property.
  - *Als je straks een list van gasten wilt printen, voldoet de `__str__()` methode in de class `Hotelgast`?*
- Programmeer de methode `__str__()`: De output is "Hotel: *naam*"

Voeg aan de klasse `Hotel` de methode **`check_in(naam, voornaam)`** toe:

- Deze methode heeft als parameters de naam en voornaam van de nieuwe gast.
- Maak in deze methode een object van de klasse `Hotelgast`.
  - Waarop moet je letten bij het importeren van de klasse `Hotelgast`???
- Controleer vooraf of deze nieuwe hotelgast reeds niet in de list aanwezig is.
  - *Voordat je dit kan doen, moet je (of Python) natuurlijk kunnen controleren wanneer 2 hotelgasten aan elkaar gelijk zijn. Pas de klasse `Hotelgast` aan.*
- Indien nog niet in het Hotel aanwezig:
  - Wijzig de property 'is\_ingecheckt' van de nieuwe hotelgast naar `True`.
  - Voeg de nieuwe hotelgast toe aan de list gasten.
  - Print in de methode de boodschap: `Correct ingecheckt: \n hotelgast`
- Indien de hotelgast wel al aanwezig was, dan print je een foutboodschap af.

Voeg aan de klasse `Hotel` de **`print_info_gasten()`** toe:

- De output van deze functie kan je afleiden uit de test klasse.

Test de dataklasse al eens uit!

Test klasse
<pre>from model.Hotel import Hotel from model.Hotelgast import Hotelgast  hotel_howest = Hotel("Howest") print(hotel_howest)  hotel_howest.check_in("Walcarius", "Stijn") hotel_howest.check_in("Laprudence", "Christophe") print("\n") hotel_howest.print_info_gasten()</pre>
<b>Terminal</b> Hotel Howest Correct ingecheckt: OK: WALCARIUS - 0 euro Correct ingecheckt: OK: LAPRUDECE - 0 euro  Dit zijn de aanwezige gasten in het hotel Howest: OK: WALCARIUS - 0 euro OK: LAPRUDECE - 0 euro



## Oef 04b

Voeg aan de klasse Hotel de **private** methode `__zoek_hotelgast(naam,voornaam)` toe.

- Deze methode heeft als parameters de naam en voornaam
- Maak in deze methode een (tijdelijk) object hotelgast aan
- Overloop de list met gasten en ga op zoek naar het (tijdelijk) object hotelgast dat hiermee overeenstemt:
  - Indien teruggevonden: **geef** het gevonden object **terug**
  - Indien niet teruggevonden: geef *None* terug

Voeg aan de klasse Hotel de methode `check_out(naam, voornaam)` toe:

- Deze methode heeft als parameters de naam en voornaam van de gast.
- Gebruik de private methode `__zoek_hotelgast()` om de bijhorende gast terug te vinden:
  - Hotelgast is teruggevonden. Controleer eerst of zijn saldo nul is. Pas dan mag de hotelgast uit de list verwijderd worden. Is het saldo nog niet nul: geef een gepaste foutmelding.
  - Indien de hotelgast niet teruggevonden wordt, print je een gepaste foutmelding af.

Test klasse
<pre>from model.Hotel import Hotel from model.Hotelgast import Hotelgast  hotel_howest = Hotel("Howest") print(hotel_howest)  hotel_howest.check_in("Walcarius", "Stijn") hotel_howest.check_in("Laprudence", "Christophe") print("\n") hotel_howest.print_info_gasten() print("\n") print("*****") hotel_howest.check_out("Walcarius", "Stijn") hotel_howest.check_out("Roobrouck", "Dieter") print("*****") print("\n") hotel_howest.print_info_gasten()</pre>
<p><b>Terminal</b></p> <pre>Hotel Howest Correct ingecheckt: OK: WALCARIUS - 0 euro Correct ingecheckt: OK: LAPRUDENCE - 0 euro  Dit zijn de aanwezige gasten in het hotel Howest: OK: WALCARIUS - 0 euro OK: LAPRUDENCE - 0 euro  ***** Correct uitgecheckt: OK: WALCARIUS - 0 euro Persoon Dieter Roobrouck is niet terug gevonden! *****  Dit zijn de aanwezige gasten in het hotel Howest: OK: LAPRUDENCE - 0 euro</pre>



## Oef 04c

Voeg aan de klasse de methode **bestel\_drank(naam,voornaam,kostprijs)** toe:

- Deze methode heeft als parameters de naam en voornaam van de gast, en de kostprijs van de bestelde drank.
- Gebruik de private methode **\_\_zoek\_hotelgast()** om de bijhorende gast terug te vinden:
  - Hotelgast is teruggevonden. Verhoog zijn saldo met de doorgegeven kostprijs.
  - Indien de hotelgast niet teruggevonden wordt, print je een gepaste foutmelding af.

Test klasse
<pre>from model.Hotel import Hotel from model.Hotelgast import Hotelgast  hotel_howest = Hotel("Howest") print(hotel_howest)  hotel_howest.check_in("Walcarius", "Stijn") hotel_howest.check_in("Laprudence", "Christophe") hotel_howest.bestel_drank("Walcarius", "Stijn", 100)  print("\n") hotel_howest.print_info_gasten() print("\n") print("*****") hotel_howest.check_out("Walcarius", "Stijn") print("*****") print("\n") hotel_howest.print_info_gasten()</pre>
Terminal
<pre>Hotel Howest Correct ingecheckt: OK: WALCARIUS - 0 euro Correct ingecheckt: OK: LAPRUDENCE - 0 euro  Dit zijn de aanwezige gasten in het hotel Howest: OK: WALCARIUS - 100 euro OK: LAPRUDENCE - 0 euro ***** Gast heeft nog openstaand saldo staan en is daarom nog niet uitgecheckt: OK: WALCARIUS - 100 euro *****  Dit zijn de aanwezige gasten in het hotel Howest: OK: WALCARIUS - 100 euro OK: LAPRUDENCE - 0 euro</pre>

Voeg aan de klasse de methode **vereffen\_saldo\_gast(naam,voornaam)** toe:

- Deze methode heeft als parameters de naam en voornaam van de gast.
- Gebruik de private methode **\_\_zoek\_hotelgast()** om de bijhorende gast terug te vinden:
  - Hotelgast is teruggevonden. Zet het saldo op nul.
  - Indien de hotelgast niet teruggevonden wordt, print je een gepaste foutmelding af.

Test klasse
<pre>from model.Hotel import Hotel from model.Hotelgast import Hotelgast  hotel_howest = Hotel("Howest") print(hotel_howest)  hotel_howest.check_in("Walcarius","Stijn") hotel_howest.check_in("Laprudence","Christophe") hotel_howest.bestel_drank("Walcarius","Stijn",100)  print("\n") hotel_howest.print_info_gasten() <b>print("Stijn betaalt zijn schuld")</b> <b>hotel_howest.vereffen_saldo_gast("Walcarius", "Stijn")</b> print("\n") print("*****") hotel_howest.check_out("Walcarius","Stijn") print("*****") print("\n") hotel_howest.print_info_gasten()</pre>
<p><b>Terminal</b></p> <pre>Hotel Howest Correct ingecheckt: OK: WALCARIUS - 0 euro Correct ingecheckt: OK: LAPRUDENCE - 0 euro  Dit zijn de aanwezige gasten in het hotel Howest: OK: WALCARIUS - 100 euro OK: LAPRUDENCE - 0 euro Stijn betaalt zijn schuld ***** Correct uitgecheckt: OK: WALCARIUS - 0 euro *****  Dit zijn de aanwezige gasten in het hotel Howest: OK: LAPRUDENCE - 0 euro</pre>

## Uitbreiding – Eigen onderzoek

### Oefeningen voor thuis



#### Thuis 1

- **Stap 1: Maak een klasse Presentator.**

Van een Presentator bewaar je in private attributen:

- *Naam*
- *Voornaam*

Deze twee attributen hebben een **getter** en **setter**-property.

Schrijf de constructor met 2 parameters (naam en voornaam)

Schrijf de `__str__()` en `__repr__()` methode.

- De output zal er als volgt uitzien: "presentator: naam, voornaam"

Zorg ervoor dat je twee presentatoren met elkaar kan vergelijken, een presentator is gelijk aan een andere als én de naam én de voornaam gelijk zijn.

Test deze klasse uit in een testklasse `test_presentator.py`

Test klasse
<pre>from model.Presentator import Presentator from model.Tvprogramma import Tvprogramma  presentator1 = Presentator("Laura", "Tesoro") presentator2 = Presentator("Nathalie", "Meskens") programma1 = Tvprogramma("Belgium got talent", presentator1) programma2 = Tvprogramma("Beste kijkers", presentator2) programma2.is_actief = False  # Verkort programma3 = Tvprogramma("The Late Late Show", Presentator("James", "Corden"))  # geeft een fout programma4 = Tvprogramma("The Simpsons", "Homer Simpson") print("*** toon info ***") print(programma2) print(programma3) print(programma4) print(f"{programma2} is momenteel actief: {programma2.is_actief}") programma2.is_actief = True print(f"{programma2} is momenteel actief: {programma2.is_actief}") programma2.is_actief = "blablablabla" print(f"{programma2} is momenteel actief: {programma2.is_actief}")</pre>
Terminal
<pre>*** toon info *** TV Programma: Beste kijkers door &lt;&lt;presentator: Nathalie, Meskens&gt;&gt; TV Programma: The Late Late Show door &lt;&lt;presentator: James, Corden&gt;&gt; TV Programma: The Simpsons door &lt;&lt;ERROR&gt;&gt; TV Programma: Beste kijkers door &lt;&lt;presentator: Nathalie, Meskens&gt;&gt; is momenteel actief: False TV Programma: Beste kijkers door &lt;&lt;presentator: Nathalie, Meskens&gt;&gt; is momenteel actief: True TV Programma: Beste kijkers door &lt;&lt;presentator: Nathalie, Meskens&gt;&gt; is momenteel actief: False</pre>

- **Stap 2: Maak een klasse Tvprogramma**

Van een Tv programma bewaar je in private attributen

- de *titel*
- *presentator*
- *is\_actief* (of het al dan niet wordt uitgezonden)

Maak de publieke properties aan voor deze attributen

- De *titel* heeft enkel een **getter** property
- De *presentator* heeft een **setter** en **getter** property.
  - De setter property aanvaardt enkel een value die van het type *Presentator* is. Wordt een andere waarde doorgegeven, dan bewaar je de string "ERROR"
- *is\_actief* heeft een **setter** en **getter** property.
  - De setter property aanvaardt enkel een value *True* of *False*. (Op welk type zal je controleren)
  - Wordt er een andere waarde doorgegeven, dan bewaar je *False*.

Schrijf de constructor die 2 parameters binnenkrijgt.

- De *titel* en *presentator*.
- De constructor stelt echter wel alle 3 de attributen in. Het attribuut *is\_actief* krijgt de waarde *True* bij het aanmaken van een instantie.

Schrijf tenslotte nog een `__str__()` en `__repr__()` methode.

- Deze twee methodes geven volgende (zelfde) output.
  - TV Programma: *titel* door `<<presentator>>`

Test de klasse associatie tussen *Presentator* en *Tvprogramma* uit in een testklasse

`test_tvprogramma.py`

Test klasse
<pre> from model.Presentator import Presentator from model.Tvprogramma import Tvprogramma  presentator1 = Presentator("Laura", "Tesoro") presentator2 = Presentator("Nathalie", "Meskens") programma1 = Tvprogramma("Belgium got talent", presentator1) programma2 = Tvprogramma("Beste kijkers", presentator2) programma2.is_actief = False  # Verkort programma3 = Tvprogramma("The Late Late Show", Presentator("James", "Corden"))  print("**toon info**") print(programma2) print(f"{programma2} is momenteel actief: {programma2.is_actief}") programma2.is_actief = True print(f"{programma2} is momenteel actief: {programma2.is_actief}") programma2.is_actief = "blablablabla" print(f"{programma2} is momenteel actief: {programma2.is_actief}") </pre>
<p><b>Terminal</b></p> <pre> **toon info** TV Programma: Beste kijkers door &lt;&lt;presentator: Nathalie, Meskens&gt;&gt; TV Programma: Beste kijkers door &lt;&lt;presentator: Nathalie, Meskens&gt;&gt; is momenteel actief: False TV Programma: Beste kijkers door &lt;&lt;presentator: Nathalie, Meskens&gt;&gt; is momenteel actief: True TV Programma: Beste kijkers door &lt;&lt;presentator: Nathalie, Meskens&gt;&gt; is momenteel actief: False </pre>



### Stap 3 Maak een klasse Zender

Van een Zender bewaar je in private attributen

- de *naam*
- de *taal* waarin ze uitzenden
- en een list met *programma's* die ze uitzenden.

Maak de correcte publieke properties aan voor de attributen

- De naam heeft een **setter** en **getter** property.
- De taal heeft een **setter** en **getter** property.
  - Er kan enkel NL / ENG / FR worden bewaard in dit attribuut.
  - Wordt er toch een andere waarde doorgegeven, dan bewaar je "ERROR".

Bij het aanmaken van een instantie van de klasse Zender geef je 2 parameters mee in de constructor.

- *Naam* en *taal*.
- Het attribuut met *programma's* wordt ingesteld als een lege lijst.

Schrijf de `__str__()` methode.

- De output zal er als volgt uitzien: "naam -> lijst met programma's "

Deze klasse bevat een extra methode **voeg\_programma\_toe(programma)**

- Deze methode krijgt als parameter een instantie van de klasse *programma* mee.
- Na controle of de parameter wel een programma is, voeg je deze toe aan de list met programma's.

Deze klasse bevat een extra methode **zoek\_afgelopen\_programmas()**.

- Deze methode heeft geen extra parameters
- Deze methode geeft een lijst terug met alle programma's die zijn afgelopen.

Deze klasse bevat een extra methode **selecteer\_willekeurig\_programma()**.

- Deze methode heeft geen extra parameters
- Deze methode geeft één willekeurig programma terug uit de lijst met programma's.

Hou, in de klasse zelf, tenslotte bij hoeveel keer er een instantie van deze klasse wordt aangemaakt.

- Voorzie een static variabele en static methode **geef\_aantal\_aangemaakte\_zenders()** om dit op te vragen.



Test dit tenslotte uit in de testklasse test\_zenders.py.

#### Test klasse

```
from model.Presentator import Presentator
from model.Tvprogramma import Tvprogramma
from model.Zender import Zender

presentator1 = Presentator("Laura", "Tesoro")
presentator2 = Presentator("Nathalie", "Meskens")

programma1 = Tvprogramma("Belgium got talent", presentator1)
programma2 = Tvprogramma("Beste kijkers", presentator2)
programma2.is_actief = False

# Verkort
programma3 = Tvprogramma("The Late Late Show", Presentator("James", "Corden"))

zender1 = Zender("VTM", "NL")
zender2 = Zender("FOX", "ENG")
zender3 = Zender("TV China", "Chinees")

zender1.voeg_programma_toe(programma1)
zender1.voeg_programma_toe(programma2)
zender2.voeg_programma_toe(programma3)
zender2.voeg_programma_toe("dit zal niet lukken")

print("*** info zenders ***")
print(zender1)
print(zender2)
print(zender3)
print(f"**** volgende programma's zijn afgelopen van {zender1.naam} ****")
print(zender1.zoek_afgelopen())
print(
    f"**** volgend programma wordt willekeurig gekozen van {zender1.naam} ****")
print(zender1.selecteer_willekeurig_programma())
print(
    f"Aantal verschillende zenders aangemaakt {Zender.geef_aantal_aangemaakte_zenders()}")
```

#### Terminal

```
*** info zenders ***
VTM (NL) ---> [TV Programma: Belgium got talent door <<presentator: Laura, Tesoro>>,
TV Programma: Beste kijkers door <<presentator: Nathalie, Meskens>>]
FOX (ENG) ---> [TV Programma: The Late Late Show door <<presentator: James, Corden>>]
TV China (ERROR) ---> []
**** volgende programma's zijn afgelopen van VTM ***
[TV Programma: Belgium got talent door <<presentator: Laura, Tesoro>>]
**** volgend programma wordt willekeurig gekozen van VTM ***
TV Programma: Beste kijkers door <<presentator: Nathalie, Meskens>>
Aantal verschillende zenders aangemaakt 3
```