Python OO Programming

Les 2b: data classes

Kristof Michiels

Wat zijn data classes?

Data classes

- Een alternatieve manier om klassen te definiëren
- Toegevoegd in Python 3.7
- Help de creatie en beheer van klassen te automatiseren
- https://docs.python.org/3/library/dataclasses.html

Hoe gebruiken?

- *Dataclass* klasse importeren uit de *dataclasses* module
- @dataclass decorator gebruiken
- We verwijderen de __init__ functie
- Instance attributen worden aangemaakt bvb als volgt: "titel: str"
- Achter de schermen zal de data class decorator de klasse herschrijven
- In dit proces worden ook __repr__ en __eq__ magic methods toegevoegd:
  - je kan object dus out of the box met elkaar vergelijken
- Het zijn Python klassen zoals elke andere die we reeds hebben gezien, maar veel beknopter geschreven

Basisvoorbeeld Data classes

```
from dataclasses import dataclass

@dataclass
class Boek:
    titel: str
    auteur: str
    lengte: int
    prijs: float

    def boekinfo(self):
        return f"{self.titel}, by {self.auteur}"

oBoek1 = Boek("Oorlog en Vrede", "Leo Tolstoj", 1225, 39)
oBoek2 = Boek("De vanger in het graan", "JD Salinger", 234, 29)
```

Basisvoorbeeld Data classes

```
print(oBoek1.titel)
print(oBoek2.auteur)

print(oBoek1)

oBoek3 = Boek("Oorlog en Vrede", "Leo Tolstoj", 1225, 39)
print(oBoek1 == oBoek3)

oBoek1.titel = "Anna Karenina"
oBoek1.lengte = 864
print(oBoek1.boekinfo())
```

Post-initialisatie gebruiken

Post-initialisatie gebruiken

- De __post_init__ method laat toe extra eigenschappen toe te voegen na het initialisatieproces
- Bvb nuttig als deze eigenschap samengesteld is of gebaseerd is op bij initalisatie gecreëerde variabelen

Voorbeeld: post-initialisatie

```
from dataclasses import dataclass

@dataclass
class Boek:
    titel: str
    auteur: str
    lengte: int
    prijs: float

    def __post_init__(self):
        self.beschrijving = f"{self.titel} door {self.auteur}, {self.lengte} pagina's"

oBoek1 = Boek("Oorlog en Vrede", "Leo Tolstoj", 1225, 39)
oBoek2 = Boek("De vanger in het graan", "JD Salinger", 234, 29)
print(oBoek1.beschrijving)
print(oBoek2.beschrijving)
```

Standaardwaarden gebruiken

Standaardwaarden gebruiken

- Je kan op eenvoudige manier standaardwaarden (= default values) meegeven bij het declareren van de instance variabelen
- Let op: attributen zonder defaultwaarden horen eerst gedefinieerd te worden
- Een andere - meer flexibele - manier om defaults toe te voegen is via de field functie:
  - "prijs: float = field(default=0.0)"
- Met default_factory kan je externe functies aanroepen en de returnwaarde als default laten gelden

Voorbeeld: standaardwaarden gebruiken

```
from dataclasses import dataclass, field
import random

def prijs_func():
    return float(random.randrange(20, 40))

@dataclass
class Boek:
    titel: str = "Geen titel"
    auteur: str = "Geen auteur"
    lengte: int = 0
    prijs: float = field(default_factory=prijs_func)
```

Voorbeeld: standaardwaarden gebruiken

```
oBoek1 = Boek()
print(oBoek1)

oBoek2 = Boek("Oorlog en Vrede", "Leo Tolstoj", 1225)
oBoek3 = Boek("De vanger in het graan", "JD Salinger", 234)
print(oBoek2)
print(oBoek3)
```

Onveranderlijke data classes

Onveranderlijke (=immutable) data classes

- Data kan niet veranderd worden van buiten de klasse
- @dataclass decorator => (frozen = True)

Voorbeeld: onveranderlijke data classes

```
from dataclasses import dataclass

@dataclass(frozen=True)
class OnveranderlijkeKlasse:
    waarde1: str = "Waarde 1"
    waarde2: int = 0

    def mijnfunctie(self, nieuwewaarde):
        self.waarde2 = nieuwewaarde


oOnveranderlijkeKlasse = OnveranderlijkeKlasse()
print(oOnveranderlijkeKlasse.waarde1)
oOnveranderlijkeKlasse.waarde1 = "Andere waarde"
print(oOnveranderlijkeKlasse.waarde1)
oOnveranderlijkeKlasse.mijnfunctie(20)
```

Python OO - les 2b - kristof.michiels01@ap.be