Python Programming - Labo 9

Doel van dit labo

Deze oefeningen zorgen ervoor dat je van start kan gaan met het ontwikkelen in object-georiënteerd Python. Je gebruikt ook eerder opgebouwde kennis van Python.

Oefening 1: drankjes en hun temperatuur

- Schrijf een klasse Drank waarvan de instances drankjes vertegenwoordigen. Elk drankje heeft twee attributen: een naam (die de drank beschrijft) en de ideale temperatuur
- Maak verschillende dranken aan en vraag naam en temperatuur op
- Pas de klasse Drank aan zodat temperatuur niet verplicht moet worden meegegeven maar zorg voor een defaultwaarde van 20 graden Celsius
- Maak verschillende dranken aan ga na of de standaardwaarde inderdaad wordt gerespecteerd
- Maak een nieuwe klasse LogFile aan die geïnitialiseerd wordt met een bestandsnaam
- Binnen de __init__ open je het bestand om naar te schrijven en ken je het toe aan een instance attribuut *bestand*
- Maak een method aan via dewelke je een meegegeven string wegschrijft naar het bestand
- Maak tenslotte een log-method aan in de klasse Drank waarmee je naam en temperatuur laat loggen naar een bepaald bestand

Oefening 2: boeken, kasten en schappen

- Schrijf een Boek klasse die je boeken laat aanmaken met een titel, auteur en prijs

- Maak daarna een Kast klasse, waarin je één of meerdere boeken kan plaatsen met een voeg_boek_toe method
- Tenslotte voeg je een totale_prijs method toe die de totaalprijs van alle boeken in de kast zal teruggeven
- Schrijf een method bevat_boek, die een string als argument neemt en True of False teruggeeft, afhankelijk van of een boek met deze titel ergens in de kast aanwezig is
- Pas je Boek klasse aan met een attribuut dikte, waarin je opslaat hoe veel ruimte een boek inneemt
- Pas de klasse Kast aan met een attribuut ruimte
- Zorg ervoor dat als een boek wordt toegevoegd en de kast vol zit (dus totale diktes van de boeken de ruimte overschrijdt) dit niet meer lukt. De gebruiker wordt hiervan op de hoogte gebracht
- Pas het geheel nog aan zodat een Kast kan bestaan uit één of meer schappen (klasse Schap)
- Je voegt boeken dan toe aan een schap
- Introduceer ruimte op schapniveau zodat een schap kan vol zitten en je de ruimte van een kast bepaalt als de som van die van de schappen

Oefening 3: personen in een populatie

- Maak een klasse Persoon aan
- Voorzie ze van een klasse attribuut populatie die toeneemt elke keer je een nieuwe persoon instantieert
- Maak 5 personen aan en check dan of populatie op 5 staat
- Gebruik de __del__ method om de populatie met eentje te laten afnemen wanneer een instance van Persoon wordt verwijderd
- Zoek op en test uit hoe je een object instance kunt verwijderen
- Zoek op en ga na wat garbage collection is in Python en hoe het werkt: http://mng.bz/nP2a

Oefening 4: transacties

- Maak een klasse Transactie, waarbij elke object instance een storting of een opname van een bankrekening vertegenwoordigt
- Wanneer je een Transactie-instance creëert geef je een positief getal voor een storting en een negatief getal voor een opname
- Maak een klasse attribuut balans aan om de balans in de gaten te houden van alle transacties
- De balans zou op elk moment de som van stortingen en opnames moeten bevatten

Oefening 5: enveloppen

- Schrijf een klasse Envelop, met twee attributen: gewicht (een float, in gram) en is_verstuurd (een boolean, met False als default)
- Je maakt drie methods:
  - *verstuur*, die is_verstuurd op True zet. Maar enkel indien de envelop voldoende gefrankeerd is
  - *frankeer*, die portokosten als argument heeft
  - *portokosten_nodig*, die meegeeft hoeveel portokosten de envelop nodig heeft. Je zorgt ervoor dat dit automatisch resulteert in gewicht van de enveloppe x 10. Deze method moet worden uitgevoerd voor de envelop kan gefrankeerd worden. Maak hiervoor een instance variabele aan.
- Schrijf een klasse GroteEnvelop die op dezelfde manier werkt als Envelop (! overerving). Enkel is de portokost hier 15 x het gewicht

Oefening 6: mobiele toestellen

- Maak een klasse MobielToestel aan die een mobiel telefoontoestel vertegenwoordigt
- Zorg voor een kies_nummer method die een nummer voor je oproept. Als resultaat wordt een gepaste string teruggegeven in hoofdletters.
- Maak een subklasse SmartPhone die de method van de base klasse gebruikt maar een eigen method open_app heeft
- Maak een subklasse iPhone aan die zijn eigen open_app method heeft, maar ook een eigen kies_nummer method die de kies_nummer method van de base

klasse oproept, maar een de string teruggeeft in lowercase

Oefening 7: brood

- Beschrijf een klasse Brood (wit brood)
- Zorg voor een method eet_sneetjes waarbij je een integer met het aantal te eten sneetjes meegeeft
- Wat krijg je terug van deze method? Een dictionary (dict) waarin je enkele voedingsstatistieken (energie, eiwit, koolhydraten, suikers, vet) opslaat en uitrekent voor het aantal sneetjes dat werd gegeten
- Je vindt de nodige informatie hier: https://www.brood.net/gezondheid/voedingswaarde/
- Maak vervolgens 2 nieuwe klasses aan (via overerving): VolkorenBrood en ZuurdesemBrood
- Elke klasse gebruikt dezelfde eet_sneetjes method, maar met aangepaste voedingsinformatie

Oefening 8: De dierentuin 2.0

- Bestudeer de oefeningen uit het vorige labo mbt de dieren in de dierentuin
- In plaats van elke dierenklasse rechtstreeks te laten erven van de klasse Dier, maak je nieuwe klasses aan: NulpotigDier, TweepotigDier en VierpotigDier
- Deze klasses erven van klasse Dier en bepalen het aantal poten van elke instance
- Pas nu klasses Wolf, Schaap, Slang en Papegaai zo aan dat elke klasse erft van deze nieuwe klasses ipv rechtstreeks van Dier
- Welke gevolgen zijn er voor je method beschrijvingen?
- Werken we met een klasse attribuut: aantal_poten
- Hebben we een __init__ method nodig in elke subklasse, of volstaat Dier.__init__?
- De __str__ method van elke klasse hoort behalve de string die we nu teruggeven ook het dierengeluid dat elk dier maakt mee te geven. Dus bij schaap: Baa — wit schaap, 4 poten

- Hoe kan je overerving gebruiken om zoveel mogelijk code hergebruik te hebben?

Oefening 9: De dierentuin 2.0 - kooien

- Bestudeer de oefeningen uit het vorige labo mbt de dieren in de dierentuin
- Er zijn momenteel geen beperkingen qua aantal dieren per kooi
- Je beperkt het aantal dieren per kooi en maakt een GroteKooi klasse aan (zoals bij de hoorntjes)
- Je gebruikt hierbij hoeveel ruimte een dier nodig heeft en zorgt dat dat niet groter (voor het totaal aantal dieren dat in de kooi zit) is dan een bepaalde kooi ruimte biedt
- Gebruik hiervoor een ruimte_nodig attribuut bij elk dier en een ruimte_beschikbaar attribuut bij Kooi/GroteKooi
- Definieer een dictionary waarin je beschrijft welke dieren bij welke andere dieren mogen zitten. Keys: klasses en values: list van klasses die compatibel zijn met dat bepaalde dier
- Wanneer je een dier toevoegt aan een kooi check je voor compatibiliteit. Wanneer dat niet het geval is, zorg je ervoor dat het dier niet wordt toegevoegd

Oefening 10: De dierentuin 2.0 - dierentuinen

- We blijven bij de dieren in de dierentuin en passen nu Dierentuin aan
- Pas dieren_per_kleur aan zodat het verschillende kleuren tegelijk als argument kan nemen
- Dieren die voldoen aan één van de kleuren worden teruggegeven.
- Zorg voor twee instances van de Dierentuin klasse die elk een afzonderlijke dierentuin vertegenwoordigen
- Zorg voor een manier om dieren te transfereren van de ene dierentuin naar de andere. Implementeer hiervoor een verhuis_dier method met als attributen ontvangende_dierentuin en een subklasse van Dier als argumenten

- Het eerste dier van het meegegeven type wordt verwijderd uit de dierentuin waarop we de method hebben aangeroepen en toegevoegd aan de eerste kooi van de ontvangende dierentuin
- Combineer de dieren_met_kleur en dieren_met_aantal_poten methods tot een enkele method geef_dieren. Maak gebruik van keyword argumenten (kleur, poten) zodat de method zelf kan uitmaken welke query moet gemaakt worden