Eindopdracht Python/ODOO

Galileo IT Services

Esmond Blaauw

# Voorwoord

Deze opdracht is gemaakt in het kader van mijn leerproces in Python en ODOO. Ik wil graag Frans Beskers, Robin Bassie, mijn Python-ODOO teamgenoten en de rest van mijn enthousiaste collega’s van Galileo IT Services bedanken voor hun steun en vertrouwen.

Inleiding

Samenvatting

Hoofdstukindeling

# Opdracht

Eindopdracht Python - Persoonlijk Uitgavenbeheer

**Beschrijving:**

Je gaat een programma maken waarmee gebruikers hun uitgaven kunnen beheren. Het programma heet Persoonlijk Uitgavenbeheer en stelt gebruikers in staat om:

 Uitgaven toe te voegen.

 Een overzicht te zien van alle uitgaven.

 Uitgaven te filteren op een of meerdere categorieën.

 Het gemiddelde bedrag van uitgaven tussen twee datums te berekenen.

 Uitgaven te filteren op twee of meer categorieën.

 Een specifieke uitgave te verwijderen.

 Uitgaven op te slaan in een .txt-bestand en later weer in te laden.

**Wat moet het programma kunnen?**

Het programma moet de volgende functionaliteiten bevatten:

**1. Uitgaven toevoegen**:

Laat de gebruiker een datum, bedrag, categorie en omschrijving invoeren. Controleer of de invoer klopt, zoals:

* Of de datum in het juiste formaat is (bijvoorbeeld YYYY-MM-DD).
* Of het bedrag een geldig getal is.

**2. Alle uitgaven bekijken**:

Toon een overzicht van alle uitgaven. Zorg dat het overzicht duidelijk is, en dat elke kostenpost geindexeerd is.

**3. Filteren op categorieën**:

Laat de gebruiker één of meer categorieën invoeren (gescheiden door komma’s). Toon alleen de uitgaven die passen bij de opgegeven categorieën. Als er geen uitgaven zijn, laat dit dan weten.

**4. Gemiddelde berekenen tussen twee datums:**Laat de gebruiker een begin- en einddatum invoeren. Bereken het gemiddelde bedrag van alle uitgaven tussen deze datums. Als er geen uitgaven zijn, laat dit ook weten.

**5. Filteren op meerdere categorieën:**

Laat de gebruiker twee of meer categorieën invoeren. Toon alle uitgaven die passen bij één van deze categorieën. Als er geen uitgaven zijn, geef dan een melding.

**6. Uitgaven verwijderen:**

Laat de gebruiker een specifieke uitgave kiezen om te verwijderen. Toon bijvoorbeeld een lijst met nummers of ID’s, zodat de gebruiker makkelijk kan kiezen.

**7.** **Opslaan en laden**:

 **Opslaan**: Sla de uitgaven op in een .txt-bestand, zodat de gegevens bewaard blijven.

 **Laden**: Wanneer het programma opnieuw start, laad de opgeslagen uitgaven automatisch in.

 **8. Menu:**

Zorg voor een simpel menu waarmee de gebruiker makkelijk kan kiezen wat hij/zij wil doen, bijvoorbeeld:

1: Uitgaven toevoegen

2: Uitgaven bekijken

3: Filteren op categorieën

4: Gemiddelde berekenen tussen twee datums

5: Filteren op meerdere categorieën

6: Uitgave verwijderen

7: Opslaan en laden

8: Afsluiten

**Wat verwachten we van jou?**

1. **Duidelijke code:**Zorg dat je code netjes is en makkelijk te begrijpen. Gebruik duidelijke namen voor functies en variabelen.
2. **Foutmeldingen toevoegen:** Zorg dat je programma netjes reageert op verkeerde invoer (bijvoorbeeld een foutieve datum of een bedrag dat geen nummer is).
3. **Test het programma**: Controleer of alle functies goed werken.

**Wat lever je in?**

1.      Het werkende Python-programma.

2.      Een korte uitleg (bijvoorbeeld in een README-bestand) waarin je beschrijft

     - Hoe je het programma hebt gemaakt.

     - Welke problemen je bent tegengekomen en hoe je die hebt opgelost.

     - Hoe het programma gebruikt kan worden.

**Stappen om het programma te maken**

1. **Uitgaven toevoegen**: Laat de gebruiker een datum, bedrag, categorie en omschrijving invoeren. Controleer of de invoer geldig is. Voeg de uitgaven toe aan een lijst.
2. **Uitgaven bekijken**: Toon een overzicht van alle uitgaven. Zorg dat het overzicht netjes is, bijvoorbeeld met rijen en kolommen.
3. **Filteren op categorieën**: Laat de gebruiker één of meerdere categorieën invoeren. Toon alleen de uitgaven die bij die categorieën horen.
4. **Gemiddelde berekenen tussen twee datums**: Laat de gebruiker twee datums invoeren (begin- en einddatum). Zoek de uitgaven binnen deze periode en bereken het gemiddelde bedrag. Toon het resultaat.
5. **Filteren op meerdere categorieën**: Laat de gebruiker een lijst van categorieën invoeren. Toon alle uitgaven die passen bij één van deze categorieën.
6. **Uitgaven verwijderen**: Toon een lijst van alle uitgaven. Laat de gebruiker een specifieke uitgave kiezen om te verwijderen, bijvoorbeeld door een nummer of ID in te voeren.
7. **Opslaan en laden**:
   * **Opslaan**: Sla alle uitgaven op in een .txt-bestand.
   * **Laden**: Bij het starten van het programma, laad de uitgaven automatisch vanuit het .txt-bestand.
8. **Menu maken**: Voeg een menu toe zodat gebruikers eenvoudig kunnen kiezen wat ze willen doen.
9. **Testen**: Test het programma stap voor stap. Controleer of alle functies goed werken en verbeter waar nodig.

# Duidelijke code

De code is zo duidelijk en bondig als mogelijk geschreven.

# Foutmeldingen toevoegen

Vooral verkeerde invoer format/annotatie heb ik geprobeerd te ondervangen met het printen van foutmeldingen en breaks in de code.

# Testen programma

Het programma is getest. Specifiek: Toevoegen kosten, verwijderen kosten, overzicht en gemidddelde bedrag kostenpost.

# Programma/code zelf

Zie mail. Twee versies. De uiteindelijke versie staat hier:

### Persoonlijk Uitgavenbeheer Programma

import json

from datetime import datetime

class Expense:

    def \_\_init\_\_(self, date, amount, category, description):

        self.date = date

        self.amount = amount

        self.category = category

        self.description = description

    def \_\_repr\_\_(self):

        return f"{self.date} | {self.amount} | {self.category} | {self.description}"

class ExpenseManager:

    def \_\_init\_\_(self):

        self.expenses = []

        self.load\_expenses()

    def add\_expense(self):

        date = input("Voer de datum in (YYYY-MM-DD): ")

        amount = input("Voer het bedrag in: ")

        category = input("Voer de categorie in: ")

        description = input("Voer een omschrijving in: ")

        if not self.validate\_date(date):

            print("Ongeldige datum. Probeer opnieuw.")

            return

        if not self.validate\_amount(amount):

            print("Ongeldig bedrag. Probeer opnieuw.")

            return

        expense = Expense(date, float(amount), category, description)

        self.expenses.append(expense)

        print("Deze uitgave is toegevoegd.")

    def view\_expenses(self):

        if not self.expenses:

            print("Er zijn geen uitgaven om weer te geven.")

            return

        for idx, expense in enumerate(self.expenses):

            print(f"{idx + 1}: {expense}")

    def filter\_by\_category(self):

        categories = input("Voer de categorieën in (gescheiden door komma's): ").split(',')

        categories = [cat.strip() for cat in categories]

        filtered\_expenses = [exp for exp in self.expenses if exp.category in categories]

        if not filtered\_expenses:

            print("Geen uitgaven gevonden voor de opgegeven categorieën.")

        else:

            for expense in filtered\_expenses:

                print(expense)

    def calculate\_average\_between\_dates(self):

        start\_date = input("Voer de startdatum in (YYYY-MM-DD): ")

        end\_date = input("Voer de einddatum in (YYYY-MM-DD): ")

        if not self.validate\_date(start\_date) or not self.validate\_date(end\_date):

            print("Ongeldige datum. Probeer opnieuw.")

            return

        total = 0

        count = 0

        for exp in self.expenses:

            if start\_date <= exp.date <= end\_date:

                total += exp.amount

                count += 1

        if count == 0:

            print("Geen uitgaven gevonden tussen deze datums.")

        else:

            print(f"Gemiddeld bedrag: {total / count:.2f}")

    def remove\_expense(self):

        self.view\_expenses()

        index = int(input("Voer de index in van de uitgave die je wilt verwijderen: ")) - 1

        if 0 <= index < len(self.expenses):

            del self.expenses[index]

            print("Uitgave voor deze index is verwijderd.")

        else:

            print("Ongeldig index.")

    def save\_expenses(self):

        with open('expenses.txt', 'w') as f:

            json.dump([exp.\_\_dict\_\_ for exp in self.expenses], f)

        print("Uitgaven opgeslagen.")

    def load\_expenses(self):

        try:

            with open('expenses.txt', 'r') as f:

                expenses\_data = json.load(f)

                self.expenses = [Expense(\*\*item) for item in expenses\_data]

        except FileNotFoundError:

            self.expenses = []

    @staticmethod

    def validate\_date(date\_str):

        try:

            datetime.strptime(date\_str, '%Y-%m-%d')

            return True

        except ValueError:

            return False

    @staticmethod

    def validate\_amount(amount\_str):

        try:

            float(amount\_str)

            return True

        except ValueError:

            return False

def main():

    manager = ExpenseManager()

    while True:

        print("\nKies een optie:")

        print("1: Uitgaven toevoegen")

        print("2: Uitgaven bekijken")

        print("3: Filteren op categorieën")

        print("4: Gemiddelde berekenen tussen twee datums")

        print("5: Uitgave verwijderen")

        print("6: Opslaan")

        print("7: Afsluiten")

        choice = input("Voer je keuze in: ")

        if choice == '1':#Toevoegen kostenpost

            manager.add\_expense()

        elif choice == '2': # Overzicht kostenposten

            manager.view\_expenses()

        elif choice == '3': # Filter per 1 of meerdere categorieën

            manager.filter\_by\_category()

        elif choice == '4': # Bereken gemiddelde bedrag tussen 2 datums

            manager.calculate\_average\_between\_dates()

        elif choice == '5': # Verwijderen kostenpost

            manager.remove\_expense()

        elif choice == '6': # Opslaan kostenpost

            manager.save\_expenses()

        elif choice == '7': # Aflsuiten programma

            manager.save\_expenses()

            print("Eindigen Persoonlijke Uitgavenbeheer programma!")

            break

        else:

            print("Ongeldige invoer. Probeer opnieuw.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# Korte uitleg/README bestand

Het is een voorwaarde om een bestaand text bestand vooraf aanwezig te hebben met de naam expenses.txt. Als dit niet het geval is wordt er wel netjes de foutmelding gemaakt en wordt een nieuw bestand expenses.txt aangemaakt.

Wil deze naam anders zijn moet dat worden gehardcode maar dat is niet moeilijk.

# Hoe is het programma gemaakt

Meerdere keren ben ik begonnen het programma te schrijven. Eerte paar keren van scratch. Daarna met behulp van andere soort gelijke programma’s van GitHub.

Later ook ChatGPT gebruikt en Co-Pilot geintegreerd in mijn VisualCode. Gedurende die tijd hebben Python/Odoo collega’s meerdere keren kort geholpen. In een latere fase heft Paul intensiever geholpen.

# Opgelopen en opgeloste problemen

Door oude instelling betreft programmeren en nieuwe programmeertaal en de manier van werken ermee ben ik vaak tegen issues aangelopen. De hoge frequentie van foutmeldingen binnen VisualStudio maakten mij onzeker en gedemotiveerd.   
  
Specifiek zijn de problemen geweest:

Gebrek aan ervaring met syntax van Python.

Tekortkomen van ervaring hoe snel foutmeldingen van VisualStudio te lossen.