

LU02.A10 - Immutable Dataclass



Implementieren Sie eine Funktion, die eine neue Instanz einer Dataclass erstellt, anstatt die Originalinstanz zu verändern, und nutzen Sie dabei die Prinzipien der funktionalen Programmierung.

Aufgabenstellung

Im Rahmen der funktionalen Programmierung ist es von entscheidender Bedeutung, Datenstrukturen unveränderlich (immutable) zu gestalten. Indem wir immutable Dataclasses verwenden, vermeiden wir unbeabsichtigte Seiteneffekte und sorgen dafür, dass unsere Datenstrukturen nur durch explizite, kontrollierte Operationen verändert werden. In dieser Aufgabe werden Sie lernen, wie man eine Dataclass verwendet, um eine unveränderliche (immutable) Datenstruktur zu erstellen, und wie man diese im Kontext der funktionalen Programmierung manipuliert.

1. Definieren Sie eine `@dataclass(frozen=True)` namens `Car` mit den Attributen `brand (str)`, `mileage (int)` und `service_dates (List[str])`.
2. Implementieren Sie eine Funktion `add_mileage(car: Car, distance: int) → Car`, die eine neue Instanz der Dataclass `Car` erstellt, mit einer aktualisierten `mileage` und einer neuen Service-Datum, wenn die Laufleistung über 10.000 km steigt.
 1. Die neue Instanz soll das aktualisierte Datum in der Liste `service_dates` speichern. Das Datum kann dabei ein fester Wert wie `2024-08-28` sein (für Testzwecke).
3. Implementieren Sie eine zweite Funktion `check_service(car: Car) → bool`, die prüft, ob die Laufleistung über 10.000 km liegt und entsprechend `True` oder `False` zurückgibt.
4. Nutzen Sie die Funktionen, um mehrere Änderungen an einer Instanz von `Car` durchzuführen, und drucken Sie die Liste der Service-Daten am Ende aus.

Code Vorlage

```
from car import Car

def add_mileage(car, distance):
    """
    Returns a new Car instance with updated mileage and possibly an updated
    service date.
    """
    new_mileage = car.mileage + distance
    new_service_dates = car.service_dates[:]

    if new_mileage > 10000 and (car.mileage <= 10000):
        new_service_dates.append('2024-08-28')

    return Car(brand=car.brand, mileage=new_mileage,
               service_dates=new_service_dates)
```

```
def check_service(car):
    """
    Checks if the car's mileage exceeds 10,000 km.
    """
    return car.mileage > 10000

if __name__ == '__main__':
    my_car = Car(brand='Toyota', mileage=9500)
    print(f'Vor der Fahrt: {my_car}')

    my_car = add_mileage(my_car, 600) # Sollte den Service hinzufügen
    print(f'Nach der ersten Fahrt: {my_car}')

    if check_service(my_car):
        print('Service benötigt!')

    my_car = add_mileage(my_car, 100) # Keine Änderung bei Service-Daten
    print(f'Nach der zweiten Fahrt: {my_car}')
    print(f'Service-Daten: {my_car.service_dates}')
```

Schritt für Schritt

1. Definieren Sie die immutable Dataclass Car in einer separaten Datei car.py.
2. Implementieren Sie die Funktion add_mileage, die eine neue Instanz zurückgibt, wenn die Laufleistung aktualisiert wird, und fügt gegebenenfalls ein Service-Datum hinzu.
3. Implementieren Sie die Funktion check_service, die die Laufleistung überprüft.
4. Führen Sie mehrere Funktionsaufrufe durch, um die Änderungen zu testen und die Liste der Service-Daten zu überprüfen.

Lösung

```
# car.py
from dataclasses import dataclass, field
from typing import List

@dataclass(frozen=True)
class Car:
    brand: str
    mileage: int
    service_dates: List[str] = field(default_factory=list)
```

```
# main.py
from car import Car

def add_mileage(car, distance):
    """
```

```
Returns a new Car instance with updated mileage and possibly an updated
service date.
"""
new_mileage = car.mileage + distance
new_service_dates = car.service_dates[:]

if new_mileage > 10000 and (car.mileage <= 10000):
    new_service_dates.append('2024-08-28')

return Car(brand=car.brand, mileage=new_mileage,
service_dates=new_service_dates)

def check_service(car):
    """
    Checks if the car's mileage exceeds 10,000 km.
    """
    return car.mileage > 10000

if __name__ == '__main__':
    my_car = Car(brand='Toyota', mileage=9500)
    print(f'Vor der Fahrt: {my_car}')

    my_car = add_mileage(my_car, 600) # Sollte den Service hinzufügen
    print(f'Nach der ersten Fahrt: {my_car}')

    if check_service(my_car):
        print('Service benötigt!')

    my_car = add_mileage(my_car, 100) # Keine Änderung bei Service-Daten
    print(f'Nach der zweiten Fahrt: {my_car}')
    print(f'Service-Daten: {my_car.service_dates}')
```



© Kevin Maurizi

From:
<https://wiki.bzz.ch/> - BZZ - Modulwiki

Permanent link:
<https://wiki.bzz.ch/modul/m323/learningunits/lu02/aufgaben/dataclass1?rev=1724837420>

Last update: 2024/08/28 11:30

