Appendix: ABCs und versteckte Methoden

Julian Huber & Matthias Panny

Vererbung mit verstecken Methoden und Attributen bei ABCs

- Herausforderungen bei Vererben von `_<protected>` und `__<private>` Methoden und Attributen

```python
from abc import ABC, abstractmethod

# Abstrakte Basisklasse für fahrbare Entitäten
class Driveable(ABC):
    @abstractmethod
    def __drive(self):
        pass

# Abstrakte Basisklasse für Roboter, die sowohl fahren als auch fliegen können
class Robot(Driveable):
    def __init__(self, name):
        self.name = False
        self.__is_flying = False
        self.__is_driving = False

    def __drive(self):
        if not self.is_flying:
            self.__is_driving = True
            self.__is_flying = False
            print(f"{self.name} fährt.")

robi_1 = Robot("Robi")
"""
>>> TypeError: Can't instantiate abstract class Robot
without an implementation for abstract method '_Driveable__drive'
"""
```

Vererbung mit verstecken Methoden und Attributen bei ABCs

- Problem: Obwohl wir `__drive()` in der `Robot`-Klasse implementiert haben, können wir kein Objekt nicht instanziieren
- Der Interpreter löst die private Eigenschaft, indem `__` zu `Name Mangling` führt. Das bedeutet, dass die Methoden nicht mehr von außen aufgerufen werden können, da sie nicht mehr unter dem ursprünglichen Namen zu finden sind
- Hierdurch funktioniert es nicht mehr die Methoden mit gleichem Namen in den Subklassen zu implementieren

Vererbung mit verstecken Methoden und Attributen bei ABCs

- Die Methode ist nun unter `_<SuperClass>__<method_name>` zu finden. In diesem Fall `_Driveable__drive()`

```
from abc import ABC, abstractmethod

# Abstrakte Basisklasse für fahrbare Entitäten
class Driveable(ABC):
    @abstractmethod
    def __drive(self):
        pass

# Abstrakte Basisklasse für Roboter, die sowohl fahren als auch fliegen können
class Robot(Driveable):
    def __init__(self, name):
        self.name = False
        self.__is_flying = False
        self.__is_driving = False

    def _Driveable__drive(self):
        if not self.is_flying:
            self.__is_driving = True
            self.__is_flying = False
            print(f"{self.name} fährt.")

robi_1 = Robot("Robi")
```