OOP 2 – KLASSEN 1

1. Kompetenzen

- Ich weiß, was eine Klasse ist und wozu sie benötigt wird.
- Ich weiß, wie man mit einer Klasse ein Objekt mit seinen Attributen und Aktionen beschreiben kann.
- Ich weiß, was ein Konstruktor ist und wofür er benötigt wird.
- Ich kann den Unterschied zwischen prozeduraler und objektorientierter Programmierung erklären und die Vorteile der objektorientierten Programmierung aufzählen.
- Ich kenne die Syntax, um in Python eine Klasse zu definieren.
- Ich kann Objekte in Python anlegen und benutzen.
- Ich kann kleine Anwendungen mit einer Klasse und Objekten schreiben.

2. Einleitung – prozedurale Programmierung

Kommen wir wieder zum Objekt „Auto“ zurück. Wir betrachten das Auto nun als Objekt mit den Attributen Tankfüllung (in Liter), Kilometerstand (in Kilometer) und Benzinverbrauch (in Liter pro 100km). Das Objekt kann folgende Aktionen ausführen: Fahren, Tanken.

Wie können diese Attribute und Aktionen nun in einer Programmiersprache wie Python definiert werden?

Attribute können in Python mit …… erzeugt werden. Die Aktionen werden mit …… in Python implementiert.

Schreibe nun ein kleines Programm (Auto_prozedural.py), dass die Attribute und Aktionen des Autos (s. oben) programmiert.

- Überlege, welche Variablen und Funktionen du dazu benötigst. Definiere die Variablen zu Beginn des Programms und definiere leere Funktionen (mit dem Befehl `pass`) mit den nötigen Parametern.
- Welche Werte erhalten die Attribute zu Beginn des Programms? Der Benzinverbrauch wird auf 6,2 Liter pro 100km (6,2 l/100km) gesetzt.
- Überlege auch, wie viele und welche Parameter die Funktionen brauchen. Trage die Parameter in die Tabelle unten ein.[1] **Wie kann man in einer Funktion auf die globalen Variablen zugreifen?**
- Programmiere eine Funktion, die die Attribute des Objektes anzeigen kann. Nutze diese Funktion, um nach jeder Aktion den Zustand des Objektes (Auto) anzuzeigen (s. rechts).
- Das Objekt führt dann folgende Operationen aus:
  - Anzeige der Attribute
  - Tanken (60 Liter)
  - Anzeige der Attribute
  - Fahren (120km)
  - Anzeige der Attribute
  - Fahren (50km)

```
------------------------------------
Kilometerstand: 0
Tankfüllung: 0
Benzinverbrauch: 6.2
------------------------------------
Aktion: Tanken 60
------------------------------------
Kilometerstand: 0
Tankfüllung: 60
Benzinverbrauch: 6.2
------------------------------------
Aktion: Fahren 120
------------------------------------
Kilometerstand: 120
Tankfüllung: 52.56
Benzinverbrauch: 6.2
------------------------------------
Aktion: Fahren 50
------------------------------------
Kilometerstand: 170
Tankfüllung: 49.46
Benzinverbrauch: 6.2
------------------------------------
Aktion: Fahren 250
------------------------------------
Kilometerstand: 420
Tankfüllung: 33.96
Benzinverbrauch: 6.2
------------------------------------
Aktion: Tanken 20
------------------------------------
Kilometerstand: 420
Tankfüllung: 53.96
Benzinverbrauch: 6.2
------------------------------------
```

Bearbeite dein Programm, bis du dieses Resultat erhältst!!!

[1] Die Attribute eines Objektes, die in einer Aktion benötigt werden, werden auch als Parameter an die Funktion übertragen.

- Anzeige der Attribute
- Fahren (250km)
- Anzeige der Attribute
- Tanken (20 Liter)
- Anzeige der Attribute

Das Resultat der Ausführung wird in der rechten Abbildung gezeigt.

Aktion	Anzahl Parameter	Parameter[1]
Fahren		
Tanken		
Anzeigen		

3. Überleitung zu den Klassen

Stellen wir uns nun vor, wir benötigen ein zweites Auto in unserem Programm. Was müssen wir nun tun? Müssen wir den kompletten Kode duplizieren? Können wir Teile wiederverwerten, ohne sie zu kopieren?

.....

.....

.....

.....

.....

.....

Im obigen Beispiel werden die Attribute (Variablen) durch das Schlüsselwort `global` mit den Aktionen (Funktionen) verbunden. Das hat einen großen Nachteil: Wenn man im Programm ein zweites Auto (Objekt) definieren möchte, dann muss man alle Attributvariablen verdoppeln und einen neuen Namen geben, in dem man zum Beispiel ein Suffix 1 und einen Suffix 2 anhängt (`auto_kilometerstand` -> `auto1_kilometerstand` und `auto1_kilometerstand`). Auch alle Funktionen müssen kopiert und mit einem Suffix versehen werden.[2] Das ist sehr umständlich.

Das Kopieren hat aber noch einen zweiten Nachteil: Wenn in der Funktionsweise des Autos etwas geändert wird oder auch hinzugefügt wird, dann muss man diese Änderung in allen Kopien der Programmierung ändern. Wenn man das nicht korrekt macht, dann funktionieren die verschiedenen Objekte des Autos nicht gleich.

Daher benötigen wir ein neues Konzept in der Programmierung. Dieses neue Konzept heißt „Klasse“. In einer Klasse werden die Attribute und Aktionen gemeinsam definiert. Die Attribute sind Variablen, die innerhalb der Klasse definiert werden. Die Aktionen sind Funktionen, die auch innerhalb der Klasse definiert werden. Die Klasse beschreibt die Eigenschaften und Aktionen der Objekte. Wir werden sehen, dass die Klasse später benutzt wird, um Objekte anzulegen.

[2] Wieso müssen die Funktionen kopiert werden?

Die Definition einer Klasse beginnt in Python mit dem Schlüsselwort `class` gefolgt vom Namen der neuen Klasse:

```
class Auto:
    def __init__(self):
        # Definition der Attribute mit Anfangswert
        self.tankfuellung = 0 # Liter
        self.kilometerstand = 0 # km
        self.benzinverbrauch = 6.2 # Liter/100km

    # Definition der Aktionen
    def Tanken( self , menge ):
        self.tankfuellung += menge
        print("Aktion: Tanken",menge)

    def Fahren( self , strecke ):
        self.kilometerstand += strecke
        self.tankfuellung -= strecke/100.0*self.benzinverbrauch
        print("Aktion: Fahren",strecke)

    def Anzeigen(self):
        print("-------------------------------------")
        print("Kilometerstand:",self.kilometerstand)
        print("Tankfüllung:",self.tankfuellung)
        print("Benzinverbrauch:",self.benzinverbrauch)
        print("-------------------------------------")
```

Die Aktionen werden als Funktionen innerhalb der Klasse definiert. Die Funktionen sind gegenüber dem Schlüsselwort `class` eingerückt.[3]

Die Aktionen besitzen alle einen neuen Parameter `self`. Den Begriff `self` kann hier mit „Sich-Selber“ übersetzen. Dieser Parameter `self` stellt das Objekt selbst dar. Es enthält die Attribute des Objektes. Um auf ein Attribut zuzugreifen benutzt man das `self`-Parameter gefolgt von dem Namen des Attributs (zum Beispiel `self.tankfuellung`).[4]

Um auf das Attribut Kilometerstand zuzugreifen benutzt man:

.....

Nun müssen die Attribute noch definiert und initialisiert werden. Das geschieht in einer besonderen Funktion der Klasse mit dem Namen `__init__`. Dies ist der sogenannte Konstruktor der Klasse.

Dieser Konstruktor wird jedes Mal aufgerufen, wenn ein Objekt erstellt wird. In dieser Funktion werden die Anfangswerte der Attribute festgelegt. Dazu wird wieder der Parameter `self` benutzt.

Gib den obigen Kode in eine Datei ein (Auto_oop.py) ein. Nun können unterhalb der Klasse die Objekte anhand der Klasse angelegt werden:

```
# Anlegen des Objektes
auto_objekt = Auto()

# Ausführung der Aktionen
auto_objekt.Anzeigen()
auto_objekt.Tanken( 60 )
auto_objekt.Anzeigen()
auto_objekt.Fahren( 120 )
auto_objekt.Anzeigen()
```

[3] Wieso?
[4] Wir werden auch sehen, dass man mit `self` auch Aktionen des Objektes aufrufen kann.

```
auto_objekt.Fahren( 50 )
auto_objekt.Anzeigen()
auto_objekt.Fahren( 250 )
auto_objekt.Anzeigen()
auto_objekt.Tanken( 20 )
auto_objekt.Anzeigen()
```

Ein Objekt wird gegründet, indem man den Name der Klasse benutzt, gefolgt von zwei Klammern (`Auto()`). Danach kann man die Aktionen ausführen, in dem man den Namen des gegründeten Objektes gefolgt von der Methode aufruft.

Wenn man nun ein zweites Objekt nötig hat, dann legt man einfach ein 2. Objekt an:

```
# Anlegen eines zweiten Objektes
auto_objekt2 = Auto()

# Ausführung der Aktionen auf dem zweiten Objekt
auto_objekt2.Anzeigen()
auto_objekt2.Tanken( 10 )
auto_objekt2.Anzeigen()
auto_objekt2.Fahren( 20 )
auto_objekt2.Anzeigen()
```

Führe das Programm aus und beobachte, dass das zweite Objekt eigene Attribute hat. Die Werte der Attribute speichert jedes Objekt selbst. Ein Objekt kann immer nur auf seine eigenen Attribute zugreifen.

4. Fragen zur Einleitung

Nenne 2 Nachteile der Objektdefinition ohne Klassen zu verwenden:

1) .....

.....

.....

2) .....

.....

.....

Erkläre, wie diese Probleme durch die Klassendefinition gelöst werden:

.....

.....

.....

.....

.....

.....

Wie nennt man die Funktion, die beim Gründen eines Objektes aufgerufen wird?

.....

Was wird in dieser Funktion gemacht?

.....

.....

.....

Umkreise den Konstruktor in der Klassendefinition im obigen Beispiel.

5. Anwendungen

1. Konstruktor

   Erstelle eine Klasse `TestKlasse`, die nur einen Konstruktor enthält. Dieser Konstruktor macht nur eine Ausgabe „Der Konstruktor wird aufgerufen“. Das Programm (Konstruktor.py) erstellt dann 3 Objekte und deutet die Gründung der Objekte an:

   ```
   print("Das erste Objekt wird angelegt.")
   objekt1 = TestKlasse()
   print("Das zweite Objekt wird angelegt.")
   objekt2 = TestKlasse()
   print("Das dritte Objekt wird angelegt.")
   objekt3 = TestKlasse()
   print("Ende")
   ```

   Teste das Programm und überprüfe die Ausgabe des Programms.

2. Person

   Die Daten zu einer Person sollen in einer Klasse gespeichert werden: Name (`str`), Geburtsdatum (`datetime`), Familienstand (`str`), Adresse (`str`).

   Folgende Operationen werden vorgesehen:

   - `AendereNamen( name )`
   - `AendereGeburtsdatum( tag , monat , jahr )`
   - `AendereFamilienstand( familienstand )`
   - `AendereAdresse( adresse )`
   - `BerechneAlter()`
   - `Anzeige()` – zur Anzeige der Personeninformation inklusive Geburtsdatum und Alter

   Definiere diese Klasse (Person.py) und teste sie mit deinen Angaben. Überlege, wie man das Alter berechnen kann. Wie können die Attribute des Objektes auf die korrekten Werte gesetzt werden?

   Info zum Klasse `datetime (import datetime from datetime:)`:
   Mit `datetime( jahr, monat, tag)` kann man ein `datetime`-Objekt mit dem Datum tag/monat/jahr erzeugen. Der Aufruf `datetime.now()` generiert ein `datetime`-Objekt für das heutige Datum.

   Teste die Berechnung des Alters mit folgenden Daten:

   - Emil (Geburtsdatum 3/01/2001)
   - Yannick (Geburtsdatum 25/08/1995)
   - Eva (Hatte gestern Geburtstag und wurde im Jahr 1998 geboren)
   - Martin (Hat heute Geburtstag und wurde im Jahr 2005 geboren)
   - Antonia (Hat morgen Geburtstag und wurde im Jahr 2000 geboren)
   - Laura (Geburtsdatum 10/10/2003)

3. Ampel

   Eine Ampel soll mit einer Klasse programmiert werden. Die Klasse hat ein Attribut, das angibt, welche Farbe angezeigt wird. Dabei sind 4 Anzeigen möglich: Rot, Grün, Orange und Orange blinken.

   Die Klasse hat 3 Aktionen: `Setze()`, `Umschalten()` und `Blinken()`. Die Aktion `Setze()` hat einen Parameter `farbe`, der dazu genutzt wird, die aktuell angezeigte Farbe einzustellen. Bei der Aktion `Umschalten()` wird jeweils die nächste Farbe angezeigt (Reihenfolge: Rot->Grün->Orange->Rot->Grün->Orange->…). Bei der Aktion `Blinken()` wird die Ampel auf Orange Blinken gesetzt. Danach sollte die Ampel beim Umschalten wieder bei Rot beginnen. Bei jeder Aktion wird der neue Zustand der Ampel angezeigt.

   Programmiere diese Klasse (Ampel.py) mit ihren Attributen und ihren Aktionen. Teste danach die Sequenz: 9xUmschalten, dann 1x Blinken, dann 2xUmschalten, Setze(Rot), dann 1x Blinken, dann 1xUmschalten.

   Die durch diese Aktionen generierte Sequenz ist: Grün, Orange, Rot, Grün, Orange, Rot, Grün, Orange, Rot, Orange Blinken, Rot, Grün, Rot, Orange Blinken, Rot.

Abgabe: Blatt + 2 Programme Einleitung + 3 Programme Anwendungen

- Auto_prozedural.py
- Auto_oop.py
- Konstruktor.py
- Person.py
- Ampel.py