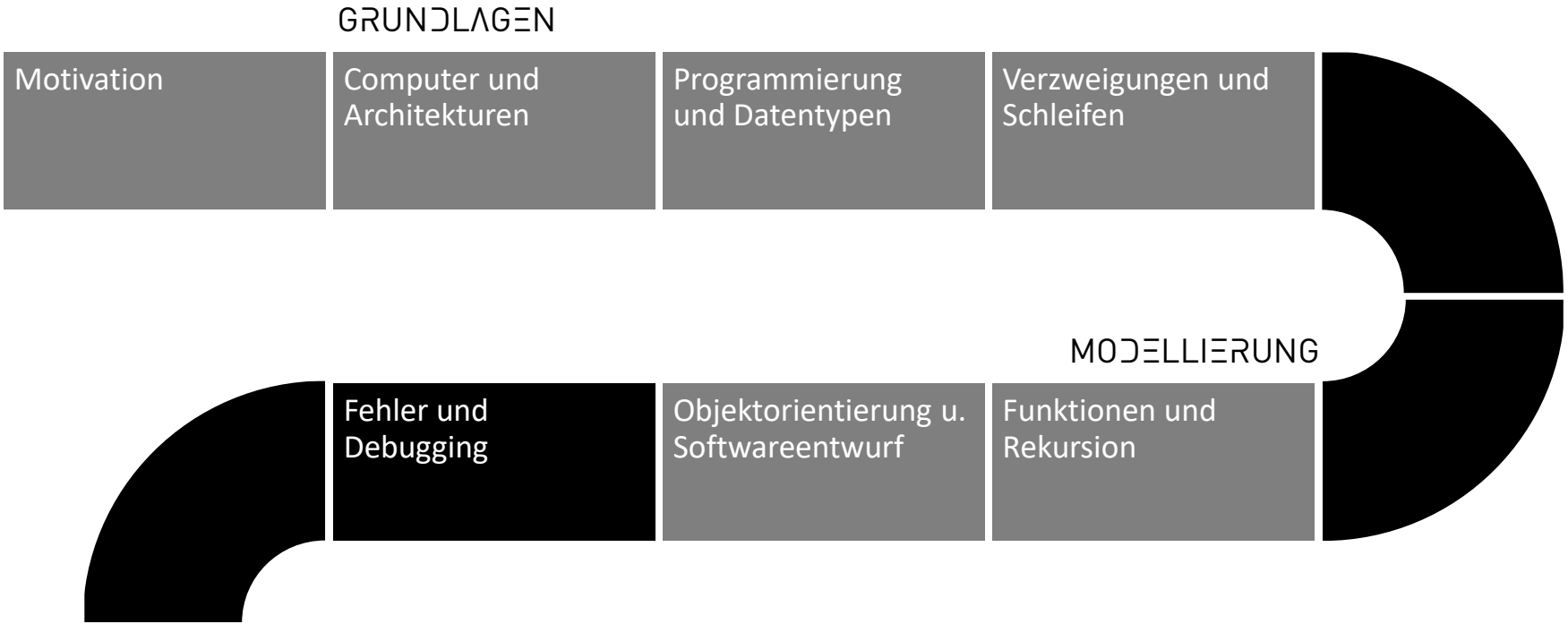
Einführung in Programmierung und Datenbanken

Joern Ploennigs

Programmierung und Datenbanken



Objektorientierung

Midjourney: Objects in the rear mirror may appear closer than they are

Zielsetzung

- Lernen der Grundidee der Objektorientierung
- Ein grundsätzliches Verständnis der Programmierung mit Objekten
  - Datentypen vs. Klassen
  - Funktionen vs. Methoden
- Einführung in erweiterte Konzepte wie z. B. Vererbung

Objektorientierte Programmierung

Objektorientierte Programmierung (OOP) ist ein Programmierparadigma das annimmt, dass ein Programm ausschließlich aus **Objekten** besteht, die miteinander kooperativ interagieren. Jedes Objekt verfügt über **Attribute** (Eigenschaften) und **Methoden**. Die Attribute definieren dabei Werte über den Zustand eines Objektes. Die Methoden definieren die möglichen Zustandsänderungen (Handlungen) eines Objektes.

Warum Objekte – Das syntaktische problem wiederholter Datenstrukturen

Objekte werden verwendet, um festzulegen wie Datenstrukturen die sich wiederholen gespeichert werden. Hierbei geht es darum, dass der Syntax der Datenstruktur eindeutig ist.

- Wächst das Programm an, so wächst auch die Menge der Variablen und Datenstrukturen
  - … zur Speicherung von Daten
  - … zur Kontrolle des Programmflusses
  - … zum Abspeichern von Zuständen
  - … zum Verarbeiten von Ein- und Ausgaben
- Dabei basieren die zugrundeliegenden Elemente meist auf sich wiederholenden Datenstrukturen.
- Analysiert man z. B. Baupläne oder Karten so verwaltet man viele Punkt-Koordinaten. Hierbei kann man unterschiedliche Koordinaten als Tupel ausdrücken.

```
punkt_1 = (54.083336, 12.108811)
punkt_2 = [12.108811, 54.083336]
```

Was ist der Syntax der Werte?

Warum Objekte – Das semantische Problem wiederholter Datenstrukturen

Objekte werden auch verwendet, um die Semantik von Werten einer Datenstruktur eindeutig zu definieren.

- Haben wir uns z. B. darauf geeinigt, dass wir ein Punkt syntaktisch durch ein Tupel repräsentieren, so ist die Bedeutung der Werte dennoch nicht bekannt.

```
punkt_1 = (54.083336, 12.108811)
punkt_2 = (12.108811, 54.083336)
```

Was ist die Semantik dieser Werte?

Warum Objekte - Das Verhaltensproblem wiederholter Datenstrukturen

Objekte werden zudem verwendet, um die Funktionen zur Verarbeitung der Datenstrukturen direkt mit dieser zu bündeln, so dass die Datenstruktur nur jene Funktionen anbietet, welche auch sinnvoll anwendbar sind.

- Definieren wir z. B. eine Funktion um die Distanz zweier Punkte zu berechnen, so kann man aufgrund der dynamischen Typisierung in Python diese ja auch auf andere Datenstrukturen anwenden, z. B. auf einer Linie. Was falsch wäre.

```python
def distanz(a, b):
    return math.sqrt((a[0]-b[0])**2 + (a[1]-b[1])**2)
```

Können die Daten mit der Funktion verarbeitet werden?

```python
punkt_1 = (54.083336, 12.108811)
punkt_2 = (12.108811, 54.083336)

distanz(punkt_1,punkt_2)
```

```python
linie_1 = [(54.08, 12.11), (54.10, 12.11)]
linie_2 = [(12.11, 54.08), (12.20, 54.10)]

distanz(linie_1, linie_2)
```

Grundlagen der Objektorientierung

- Anstatt unübersichtlich viele verstreute Datenstrukturen und Funktionen zu benutzen, gruppieren wir diese in Objekte.
- Die Struktur dieser Objekte wird in Form von **Klassen** definiert, welche eine Art Bauplan darstellt. Eine Klasse definiert:
  - welche Attribute (Variablen / Eigenschaften) ein Objekt dieser Klasse besitzt.
  - und welche Methoden (Funktionen) ein Objekt der Klasse bereit stellt

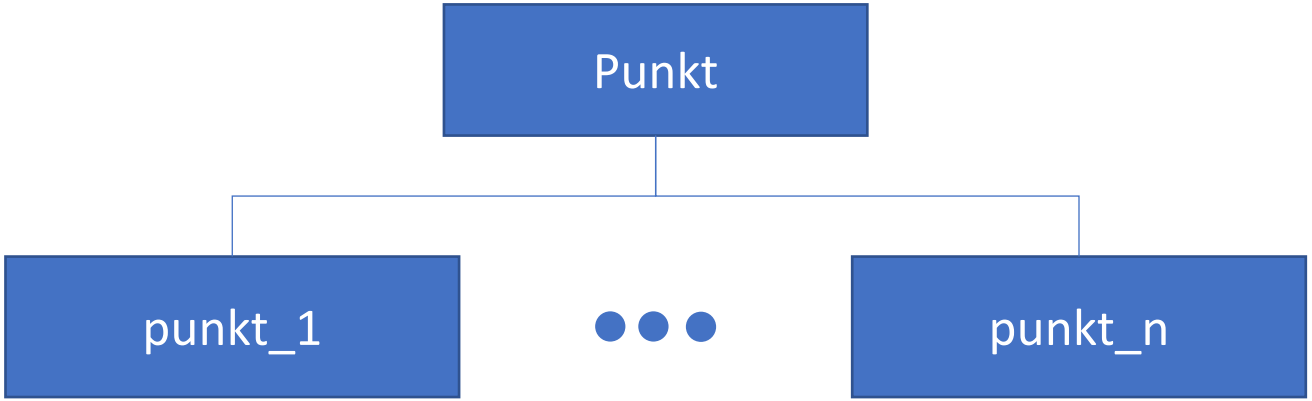
- Diese werden zusammen in einer komplexen Datenstruktur gespeichert. Jedes Objekt stellt somit einen *zusammengesetzten (komplexen) Datentypen* dar

Klassen und Instanzen

Objekte selbst sind immer Instanzen einer Klasse (die Klasse ist ja nur ein Bauplan). Eine Klasse kann beliebig viele Instanzen haben oder gar keine. Alle Instanzen sind gleich aufgebaut, besitzen aber nicht unbedingt die gleichen Werte in den Attributen.

Klassen und Instanzen

Objekte selbst sind immer Instanzen einer Klasse (die Klasse ist ja nur ein Bauplan). Eine Klasse kann beliebig viele Instanzen haben oder gar keine. Alle Instanzen sind gleich aufgebaut, besitzen aber nicht unbedingt die gleichen Werte in den Attributen.



Objektorientierung in Python

- Python ist von Grund auf objektorientiert – ein Fakt den wir bisher ignoriert haben

- Alle Datentypen in Python sind Objekte (deshalb haben sie ja auch eigene Methoden)
- Selbst definierte Klassen sind immer ein zusammengesetzter (komplexer) Datentyp

- Die Bestandteile eines Datentyp-Objektes sind:
  - Wert
  - Typ
  - Identität (ID-Nummer)

- Typ und Identität sind immer unveränderlich
- Variablen sind Referenzen auf Objekt-Instanzen

Klassen definieren

Der erste Schritt zu einem Objekt ist das Definieren einer neuen Klasse für den Typ des Objektes. Dies geschieht über das `class`-Kennwort

```
class Klassenname:
    # Klassendefinition
```

Konstruktoren mit der init-Methode

Der Konstruktor `__init__()` ist eine spezielle Methode, die festlegt wie eine neue Instanz der Klasse erzeugt wird. Er wird genutzt um Attribute initial zuzuweisen als auch Initialisierungsschritte (Tests, Berechnungen, Konfigurationen, etc.) durchzuführen.

```
class Punkt:
    # Konstruktor
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

- `self` beschreibt eine Selbst-Referenz auf die neue Instanz der Klasse
- Auf die Attribute einer Instanz kann durch den Punkt-Syntax zugegriffen werden
- `self.x` ist somit eine Referenz auf das Attribut `x` der Instanz
- `self.x = x` bedeutet dass wir den Wert der Variablen `x` dem neuen Instanzattribut `x` zuweisen (obwohl beide gleich heißen sind sie nicht die gleiche Variable!)
- Da `__init__` eine Funktion ist, wenn auch besonders, kann man auch Defaults definieren

Instanzen erzeugen

- Instanzen der Klasse werden erzeugt indem der Klassenname wie ein Funktionsname benutzt wird

- Um Instanzen zu erstellen wird der Klassenname wie ein Funktionsname benutzt (dies ruft den Konstruktor impliziert auf). Hierbei wird der self-Parameter nicht mit angegeben.

```
punkt_1 = Punkt(54.083336, 12.108811)
punkt_2 = Punkt(12.108811, 54.083336)
```

Attribute auf Instanzebene

- Instanzattribute können in jeder Instanz unterschiedlich sein.
- Ändert eine Instanz das Attribut, so wirkt sich die Änderung nicht auf andere Instanzen aus (Isolierung)
- Sie werden in dem Konstruktor `__init__` definiert.

```
class Punkt:
    # Konstruktor
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Attribute auf Klassenebene

- Klassenattribute sind Attribute welche für alle Instanzen einer Klasse den gleichen Wert haben
- Sie werden unter dem `class`-Kennwort wie eine Variable definiert
- Wichtig! Sie gelten für alle Instanzen, wenn also eine Instanz den Wert ändert, so ändert er sich in allen anderen Instanzen

```
class Punkt:
    # Attribut aller Instanzen
    einheit = „m"
```

Methoden

- Methoden werden wie Funktionen mit dem Schlüsselwort `def` definiert, auf der gleichen Ebene eingerückt wie die Klasse. Diese Methoden sind dann in allen Instanzen verfügbar
- Methoden besitzen immer `self` als ersten Parameter. Auch hier ist das eine Referenz auf aktuelle Instanz. Dadurch kann man dann auf die Attribute oder andere Methoden zugreifen
- Man kann Parametern der Methoden auch Defaults zuweisen

```
class Punkt:
    # Methode

    def distanz(self, punkt_2):

        return math.sqrt((self.x - punkt_2.x)**2 + (self.y - punkt_2.y)**2)
```

Methoden aufrufen

- Klassenmethoden sind in allen Instanzen verfügbar
- Sie werden durch den Punkt-Syntax aufgerufen, bei der man den Variablennamen links vom Punkt und den Methodennamen rechts vom Punkt aufruft

```
punkt_1.distanz()
```

- Auf gleiche Weise kann auch auf Attribute zugegriffen werden

```
print(punkt_1.x, punkt_1.y)
```

- und diese verändert werden

```
punkt_1.x = 20
```

HÖRSAALFRAGE

FRAGEN?

Midjourney: A psychedelic DJ with a question mark for a head