Python (OOP)

Kurzversion

Klasse

Eine Klasse ist ein Bauplan, um Objekte zu erzeugen.

Sie definiert die *Variablen* (*Attribute*) und die *Funktionen* (*Methoden*), welche auf die Daten angewendet werden können. In Python wird eine Klasse mit dem Schlüsselwort `class` definiert.

```
1 class Myclass:
2
3     <Anweisung 1>
4     <Anweisung 2>
5     ...
```

Objekt und Konstruktor

Ein Objekt (Synonym: Instanz) ist eine Sammlung von Variablen (Attributen) und Funktionen (Methoden), die gemäss dem Bauplan der Klasse zur Laufzeit erzeugt wird.

Die Funktion, mit der ein Objekt erzeugt wird, heisst Konstruktor.

Der Konstruktor in Python: `def __init__(self, ...):`

Es ist üblich, den ersten Parameter des Konstruktors `self` zu nennen. `self` steht für die Adresse des Objekts, die bei seiner Erzeugung zurückgegeben wird.

```
1  class Bruch:
2
3      def __init__(self, zaehler, nenner): # Konstruktor
4          self.z = zaehler # Attribut 'z'
5          self.n = nenner  # Attribut 'n'
6
7      def show(self): # Methode
8          print(f'{self.z}/{self.n}')
9
10 a = Bruch(2, 3) # Erzeuge Objekt a (a.z=2, a.n=3)
11 b = Bruch(4, 7) # Erzeuge Objekt b (b.z=4, b.n=7 )
12 a.show()        # 2/3
13 b.show()        # 4/7
```

Attribut

Ein Attribut ist ein Variable, die den Zustand der Objekte definiert.

Fasst man eine Klasse als übergeordnetes Objekt auf, so lassen sich zwei Arten von Attributen unterscheiden:

- Klassenattribut: Eine Variable, die immer nur einen Wert hat.
- Objektattribut: Eine Variable, bei der jedes Objekt einen eigenen Wert hat.

```
1  class Kreis:
2
3      pi = 3.14 # Klassenattribut
4
5      def __init__(self, radius):
6          self.r = radius # Objektattribut
7
8  k1 = Kreis(3)    # Jedes Kreisobjekt hat seinen
9  k2 = Kreis(5)    # eigenen Wert für das Attribut r
10 print(k1.r)      # -> 3
11 print(k2.r)      # -> 5
12 print(Kreis.pi) # -> 3.14
```

Methode

Eine Methode ist eine Funktion, die das Verhalten der Objekte definiert.

Wie bei Attributen unterscheidet man zwei Arten von Methoden:

- Klassenmethode: Eine Funktion, die auf den Attributen der Klasse arbeitet.
- Objektmethode: Eine Funktion, die auf den jeweiligen Attributen eines Objekts ausgeführt wird.

Objektmethoden erkennt man am ersten Parameter `self`, der in Klassenmethoden fehlt.

Eine Klassenmethode wird aufgerufen, indem man sie mit einem Punkt (`.`) an den Klassennamen bindet.

Eine Objektmethode wird aufgerufen, indem man sie mit einem Punkt (`.`) an den Objektnamen bindet.

```
1  class Punkt:
2
3      anzahl = 0 # Klassenvariable
4
5      def show_anzahl(): # Klassenmethode
6          print(Punkt.anzahl)
7
8      def __init__(self, x, y): # Konstruktor
9          self.x = x # Objektvariable
10         self.y = y # Objektvariable
11         Punkt.anzahl += 1
12
13     def show(self): # Objektmethode
14         print(f'({self.x},{self.y})')
15
16 p1 = Punkt(3, 4)
17 p2 = Punkt(-1, 5)
18 Punkt.show_anzahl() # => 2
19 p1.show()           # => (3,4)
20 p2.show()           # => (-1,5)
```

Kapselung

Kapselsung bezeichnet den kontrollierten Zugriff auf Attribute und Methoden von Klassen über eine Schnittstelle.

```
1  class Myclass:
2
3      def __init__(self, value):
4          self.x = value
5
6      def set_value(self, new_value): # Setter-Methode
7          self.x = new_value
8
9      def get_value(self): # Getter-Methode
10         return self.x
11
12 obj = Myclass(42)
13 obj.set_value(17)
14 print(obj.get_value()) # => 17
```

Polymorphie

Polymorphie (*Vielgestaltigkeit*) bedeutet, dass derselbe Bezeichner (Variablenname, Funktionsname, Operator) für verschiedene Objekte unterschiedlich definiert wird.

Überladen von Methoden

Das *Überladen* ist eine Form der Polymorphie, bei der Funktionen und Operatoren für unterschiedlicher Klassen den gleichen Namen aber eine unterschiedliche Definition haben.

Ein bekanntes Beispiel ist der Python-Operator „+“ der ganze Zahlen, Gleitkommazahlen, Zeichenketten oder Listen „addiert“.

Es gibt eine Menge von *Spezialmethoden*, mit denen Python-Funktionen und -Operatoren auf Objekte eigener Klassen angewendet werden können. Hier eine kleine Auswahl:

Name	**Aufruf mit**
`__add__(self, other)`	`obj1 + obj2`
`__lt__(self, other)`	`obj1 < obj2`
`__eq__(self, other)`	`obj1 == obj2`
`__str__(self)`	`str(obj)`
`__len__(self)`	`len(obj)`

Vererbung

Mit der Klassendefinition `class Child(Parent):` wird eine Klasse `Child` definiert, die zunächst alle Attribute und Methoden einer zuvor definierten Elternklasse `Parent` „erbt“.

Dabei können die von der Mutterklasse geerbten Attribute und Methoden ...

- *überschrieben*, also neu definiert werden, was eine weitere Form der Polymorphie ist oder
- durch zusätzliche Attribute und Methoden *erweitert* werden.

```
class Mother:

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def method1(self):
        return self.x + self.y

    def method2(self):
        return self.x - self.y
```

```
13  class Child(Mother):
14
15      def __init__(self, x, y, z): # Überschreibe
16          self.x = x               # den Konstruktor
17          self.y = y               # der Mutterklasse
18          self.z = z
19
20      def method2(self):           # Überschreibe
21          return self.y - self.x   # methode2
22
23      def method3(self):           # definiere
24          return self.y * self.z   # neue Methode
```

```
26  c = Child(3, 4, 5)
27  print(c.method1())    # => 7
28  print(c.method2())    # => 1
29  print(c.method3())    # => 20
```