

Hinweis: Diese Lernnotiz enthält einen sehr sinnvollen Vorschlag um den Lehrstoff der 2. Woche der Veranstaltung zu erlernen. Er ist gegliedert in die generellen Ziele und die Arbeitsschritte. Es ist notwendig, dass Sie die in dieser Lernnotiz genannten Videos bis zum Ende der offiziellen Vorlesungszeit (Mi 30.9, 16:00 und Mo 5.10, 16:00) durchgearbeitet haben. Zu den Vorlesungszeiten besteht die Möglichkeit in Zoom Fragen zu stellen und weitergehende Themen zu Aufgaben oder auch dem Studium zu diskutieren.

<https://hs-osnabrueck.zoom.us/my/kleuker>

Ziele VL 3

- die Begriffe Objekt, Objektvariable und Beziehungen zwischen Objekten verstehen und selbständig nutzen können
- den Zusammenhang zwischen den Begriffen Klasse und Objekt verstehen
- einfache Klassen mit Abhängigkeiten selbständig programmieren können

Arbeitsschritte VL 3

- *Laden Sie sich die folgenden Videos zuerst herunter, wenn Sie die HS-Plattform nutzen und schauen Sie sich diese an. Es ist sinnvoll die Folien danach nochmals durchzugehen.*

Folien 61 – 69: Objekte, Objektvariablen und Beziehungen

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objekt.mp4> (23:03), auch
<https://youtu.be/CHBr-SuHLXI>

Klassen in BlueJ

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Klasse1.mp4> (19:01), auch
<https://youtu.be/0-CIDK5KUBA>

Folien 70 – 92: Klassen und Objektvariablen

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Klasse2.mp4> (31:35), auch
https://youtu.be/Z8nLz-DF_j8

Folien 93 – 97: Fallstudie Klassenentwicklung

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Klasse3.mp4> (8:16), auch
<https://youtu.be/vqZ5pKLQtzo>

- Bearbeiten Sie das Quiz unter http://kleuker.iui.hs-osnabrueck.de/quiz/prog03_00134.html und merken Sie sich die oben angegebenen Lösungsbuchstaben für das Praktikum.
- Lesen Sie das Frage-und-Antworten-Dokument zur Vorlesung von der Veranstaltungs-Webseite, das spätestens am nächsten Arbeitstag online ist.
- Prüfen Sie, ob Sie die angegebenen Lernziele erreicht haben.

Ziele VL 4

- Zentrale Begriffe Konstruktor, Objektvariable, Parameter, Block und lokale Variable kennen und anwenden können.
- wissen, dass es keinen Zusammenhang zwischen Parameternamen und Objektvariablenamen geben muss, aber geben kann
- die Bedeutung von this kennen und this und this() nutzen können

Arbeitsschritte VL 4

- Laden Sie sich die folgenden Videos zuerst herunter, wenn Sie die HS-Plattform nutzen und schauen Sie sich diese an. Es ist sinnvoll die Folien danach nochmals durchzugehen.

Konstruktoren in BlueJ programmieren

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Konstruktor1.mp4> (27:41),
auch <https://youtu.be/4V3nUQzepGA>

Folien 98 – 112: Konstruktoren und Speicherdiagramme

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Konstruktor2.mp4> (14:46),
auch <https://youtu.be/PVYhcW4GoA0>

Folien 113 – 129: Varianten der Objekterzeugung

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Konstruktor3.mp4> (30:30),
auch <https://youtu.be/ySKvkiel5lc>

- Bearbeiten Sie das Quiz unter http://kleuker.iui.hs-osnabrueck.de/quiz/prog04_00135.html und merken Sie sich die oben angegebenen Lösungsbuchstaben für das Praktikum.
- Lesen Sie das Frage-und-Antworten-Dokument zur Vorlesung von der Veranstaltungs-Webseite, das spätestens am nächsten Arbeitstag online ist.
- Bearbeiten Sie Aufgabenblatt 2. Denken Sie daran, dass ich für Fragen meist kurzfristig erreichbar bin.
- Prüfen Sie, ob Sie die angegebenen Lernziele erreicht haben.

Übertragung nach Python (nicht relevant für das 1. Semester)

Folie 84

s: str

```
s = str("Moin")           # explizite Erzeugung
print(f"{s} ({type(s)})") # "Moin" (<class 'str'>)

s = "Moin"                # String-Literal (wie Java)
print(f"s + s ist {s + s}") # s + s ist MoinMoin (wie Java)

# s - s geht nicht

s = s + " " + s
print(f"{s}")             # Moin Moin
```

Die Klasse String heißt in Python str. In der print-Funktion steht vor dem String ein „f“ als Funktion. Die Funktion wird dazu genutzt, Ausdrücke, die in geschweiften Klammern stehen, auszuwerten und in den String einzubauen.

Folie 97

```
from typing import List
```

```
class Datum:
    def __init__(self, tag: int, monat: int, jahr: int) -> None:
        self.tag: int = tag
        self.monat: int = monat
        self.jahr: int = jahr

class Adresse:
    def __init__(self, strasse: str, hausnummer: str) -> None:
        self.strasse: str = strasse
        self.hausnummer: str = hausnummer
        self.postleitzahl: str = "49076"           # Default-Wert wie in Java
        self.stadt: str = "Osnabrueck"
        self.bundesland: str = "Nds"

class Einzelposten:
    def __init__(self, anzahl: int, produktname: str, einzelpreis: float) -
> None:
        self.anzahl: int = anzahl
        self.produktname: str = produktname
        self.einzelpreis: float = einzelpreis

class Bon:
    def __init__(
        self,
        unternehmen: str,
        adresse: Adresse,
        datum: Datum,
        uhrzeit: str,
        produkte: List[Einzelposten]
    ) -> None:
        self.unternehmen: str = unternehmen
        self.adresse: Adresse = adresse
        self.datum: Datum = datum
        self.uhrzeit: str = uhrzeit
        self.produkte: List[Einzelposten] = produkte
```

Die erste Besonderheit bei Python ist, dass Objektvariablen nicht einfach angegeben werden können. Objektvariablen werden in einem Konstruktor definiert. Konstruktoren haben die Form einer Funktion, die genau den Namen `__init__` haben muss. Der erste Parameter ist mit `self` immer das neu zu erstellende Objekt, das beim Aufruf nicht angegeben wird. Der Rückgabebetyp ist etwas irritierend `None`. Die zweite Besonderheit ist, dass Typen keine Standardwerte (Default-Werte) haben. Deshalb ist es sinnvoll entweder in der Parameterliste des Konstruktors immer eine Variable für den Wert anzugeben oder diesen im Konstruktor zu setzen.

Folien 100ff

```
class Datum:
    def __init__(self, tag: int, monat: int, jahr: int) -> None:
        self.tag: int = tag
        self.monat: int = monat
        self.jahr: int = jahr
```

```
# geht nicht, würde ersten Konstruktor so ersetzen!
# def __init__(self, jahr: int) -> None:
#     self.tag: int = 1
#     self.monat: int = 1
#     self.jahr: int = jahr

    @classmethod
    def nur_jahr(cls, jahr: int) -> "Datum":
        return cls(1, 1, jahr) # ruft den Konstruktor auf

d1 = Datum(15, 9, 2026)
print(f"{d1.tag:02d}.{d1.monat:02d}.{d1.jahr}") # 15.09.2026

d2 = Datum.nur_jahr(2026) # ruft Klassenmethode auf
print(f"{d2.tag:02d}.{d2.monat:02d}.{d2.jahr}") # 01.01.2026
```

In Python kann es immer nur einen Konstruktor geben! Der Programmcode zeigt einen auch in Java umsetzbaren Ansatz über eine sogenannte Klassenmethode (kommen später), die an der Annotation `@classmethod` erkennbar ist. Der erste Parameter ist dabei ein Klassenobjekt. Dieses Klassenobjekt existiert pro Klasse genau einmal und kann zur Verwaltung von Klasseneigenschaften genutzt werden. Das Klassenobjekt wird beim Aufruf nicht mit angegeben. Anders als bei Objektmethoden werden Klassenmethoden auf der Klasse aufgerufen. Das Klassenobjekt ist auch nutzbar, um den einen existierenden Konstruktor aufzurufen.

Ähnlich wie in Java gilt, wird kein Konstruktor angegeben, existiert automatisch der parameterlose Konstruktor. Die Namen `self` und `cls` sind üblich, andere Namen auch nutzbar (damit würde gezeigt werden, dass wenig Ahnung von Python vorhanden ist).

Folien 87ff, Folien 113ff
from typing import List

```
class Datum:
    def __init__(self, tag: int, monat: int, jahr: int) -> None:
        self.tag: int = tag
        self.monat: int = monat
        self.jahr: int = jahr

class Adresse:
    def __init__(self, strasse: str, hausnummer: str) -> None:
        self.strasse: str = strasse
        self.hausnummer: str = hausnummer
        self.postleitzahl: str = "49076" # Default-Wert wie in Java
        self.stadt: str = "Osnabrueck"
        self.bundesland: str = "Nds"

class Studierend:
    def __init__(
        self,
        vorname: str,
        nachname: str,
        geburtsort: str,
        geburtsland: str,
```

```
        strasse: str,
        hausnummer: str,
        tag: int,
        monat: int,
        jahr: int,
        studiengang: str,
        matrikelnummer: int,
    ) -> None:
        self.vorname: str = vorname
        self.nachname: str = nachname
        self.geburtsort: str = geburtsort
        self.geburtsjahr: int = jahr
        self.geburtsland: str = geburtsland
        self.adresse: Adresse = Adresse(strasse, hausnummer)
        self.eingeschriebenAm: Datum = Datum(tag, monat, jahr)
        self.studiengang: str = studiengang
        self.matrikelnummer: int = matrikelnummer

class Praktikum:
    def __init__(
        self,
        modul: str,
        semester: str,
        veranstaltend: str,
        mitarbeitend: str,
    ) -> None:
        self.modul: str = modul
        self.semester: str = semester
        self.veranstaltend: str = veranstaltend
        self.mitarbeitend: str = mitarbeitend
        self.termine: List[str] = []
        self.teilnehmende: List[Studierend] = []

# Einen Studierenden anlegen
s = Studierend(
    vorname="Elena",
    nachname="Li",
    geburtsort="Omsk",
    geburtsland="Russland",
    strasse="Musterstraße",
    hausnummer="12a",
    tag=15,
    monat=9,
    jahr=2023,
    studiengang="Agrarsystemtechnik",
    matrikelnummer=123456,
)

# Ein Praktikum anlegen und den Studierenden hinzufügen
p = Praktikum(
    modul="Programmieren 1",
    semester="WiSe 2026/27",
```

```
        veranstaltend="Prof. Kleuker",
        mitarbeitend="Mr. X",
    )
    p.teilnehmende.append(s)
    p.termine.append("2026-10-5")
    p.termine.append("2025-10-7")

# Ausgabe zur Kontrolle
print("=== Studierend ===")
print(f"Name: {s.vorname} {s.nachname}")
print(f"Geburtsort: {s.geburtsort}, Geburtsjahr: {s.geburtsjahr}")
print(f"Adresse: {s.adresse.strasse} {s.adresse.hausnummer}, "
      f"{s.adresse.postleitzahl} {s.adresse.stadt}")
print(f"Eingeschrieben am: {s.eingeschriebenAm.tag}."
      f"{s.eingeschriebenAm.monat}.{s.eingeschriebenAm.jahr}")
print(f"Studiengang: {s.studiengang}, Matrikel: {s.matrikelnummer}")

print("\n=== Praktikum ===")
print(f"Modul: {p.modul} ({p.semester})")
print(f"Veranstaltend: {p.veranstaltend}")
print(f"Termine: {p.termine}")
print(f"Anzahl Teilnehmende: {len(p.teilnehmende)}")

# === Studierend ===
# Name: Elena Li
# Geburtsort: Omsk, Geburtsjahr: 2023
# Adresse: Musterstraße 12a, 49076 Osnabrueck
# Eingeschrieben am: 15.9.2023
# Studiengang: Agrarsystemtechnik, Matrikel: 123456
#
# === Praktikum ===
# Modul: Programmieren 1 (WiSe 2026/27)
# Veranstalter: Prof. Kleuker
# Termine: ['2026-10-5', '2025-10-7']
# Anzahl Teilnehmende: 1
```

Im Konstruktor der Klasse Studierend wird aus einzelnen Daten ein Objekt der Klasse Adresse zusammengesetzt. Alternativ und typischer wäre es, ein Objekt der Klasse Adresse als Parameter im Konstruktor zu übergeben.

Folie 120

```
jahr: int = 2023
wert: float = 42.41
wert2: float = 42.42      # Python kennt kein double
allesToll: bool = True    # True mit großem T

print(f"jahr = {jahr} (Typ: {type(jahr)})")
print(f"wert = {wert} (Typ: {type(wert)})")
print(f"wert2 = {wert2} (Typ: {type(wert2).__name__})")
print(f"allesToll = {allesToll} (Typ: {type(allesToll)})")

# jahr = 2023 (Typ: <class 'int'>)
```

```
# wert = 42.41 (Typ: <class 'float'>)  
# wert2 = 42.42 (Typ: float)  
# allesToll = True (Typ: <class 'bool'>)
```

Python nutzt nicht die standardisierten elementaren Typen, die in fast jeder anderen Sprache genutzt werden.

Java (primitiv)	Größe	Java-Wertebereich	Python- Analogie	Bemerkung
byte	8 Bit	-128 ... 127	int	Python kennt keine festen Bitbreiten
short	16 Bit	-32 768 ... 32 767	int	
int	32 Bit	$-2^{31} \dots 2^{31}-1$	int	
long	64 Bit	$-2^{63} \dots 2^{63}-1$	int	
float	32 Bit	IEEE-754	float	Python hat nur eine Gleitkomma-Art float in Python entspricht double in Java
double	64 Bit	IEEE-754	float	
boolean	1 Bit	true / false	bool	Python hat keinen eigenen Zeichentyp
char	16 Bit	Unicode	str (Länge 1)	

Die Werte von Variablen mit elementaren Datentypen in Java entsprechen immer Objekten in Python. Der Unterschied ist nicht so drastisch, da in Python Objekte der Klassen int, float, bool und str nicht veränderbar sind (immutable). Wenn man keine speziellen Klassen in Python nutzt, gibt es keine fest vorgegebenen Bit-Breiten.

Folien 122 ff

class Datum:

```
    def __init__(self, anInt1: int, anInt2: int, anInt3: int) -> None:  
        self.tag: int = anInt1  
        self.monat: int = anInt2  
        self.jahr: int = anInt3
```

```
day: int = 6
```

```
month: int = 5
```

```
year: int = 1989
```

```
d = Datum(15, 9, 2026)
```

```
print(f"Tag: {d.tag}, Monat: {d.monat}, Jahr: {d.jahr}")
```

```
# Tag: 15, Monat: 9, Jahr: 2026
```