

Hinweis: Diese Lernnotiz enthält einen sehr sinnvollen Vorschlag um den Lehrstoff der 3. 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 7.10, 16:00 und Mo 12.10, 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 5

- Objektmethode mit Parametern schreiben, nutzen und aufrufen können
- selbstständig Hilfsklassen erstellen können, um andere Klassen und deren Objektmethode ausprobieren zu können
- die Klasse EinUndAusgabe nutzen zu können

Arbeitsschritte VL 5

- *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.*

Objektmethode in BlueJ schreiben und nutzen

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektmethode1.mp4>
(36:10), auch <https://youtu.be/8L1-VpRGHdk>

Folien 130 – 157: Objektmethode, Getter und Setter, Analyseklassen zum ausprobieren

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektmethode2.mp4>
(29:44), auch <https://youtu.be/7GQAX2R-pOg>

Laden und Nutzen der Klasse EinUndAusgabe in BlueJ

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektmethode3.mp4>
(18:33), auch <https://youtu.be/Ls0YRs7ykvw>

- Bearbeiten Sie das Quiz unter http://kleuker.iui.hs-osnabrueck.de/quiz/prog05_00235.html und merken Sie sich die oben angegebenen Lösungsbuchstaben für das Praktikum.
- Lesen Sie folgenden Text unter <https://kleuker.iui.hs-osnabrueck.de/NutzungVonKI.pdf> zur KI-Nutzung durch. Rückfragen können per E-Mail oder im Praktikum diskutiert werden.
- 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 6

- den Begriff Objektreferenz kennen und nutzen können
- den Aufbau von Speicherdiagrammen kennen und diese selbstständig erstellen können
- die Begriffe „eingeben“, „ausgeben“, „übergeben“ und „zurückgeben“ im Detail unterscheiden können und ihre Umsetzung in Programmen verstehen

Arbeitsschritte VL 6

- 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 158 – 161: Bisheriges Wissen über Klassen gesammelt

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektmethoden4.mp4>
(12:11), auch <https://youtu.be/Hx0ZTkIkr34>

Folien 162 – 164: Besonderheiten der Konsole (Ein- und Ausgabe)

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektmethoden5.mp4>
(10:14), auch <https://youtu.be/IOXMCdDUOZQ>

Folien 165 – 170: Umgang mit Objektreferenzen

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektmethoden6.mp4> (9:24),
auch <https://youtu.be/5DWIMDxUfMY>

Folien 171 – 175: Interaktionsbrett und Fallstudie zur Programmentwicklung

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektmethoden7.mp4>
(33:31), auch https://youtu.be/rZ_ptN05Ib8

Folien 176 – 180: Präzisierung typischer Begriffe und ihre Umsetzung

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektmethoden8.mp4>
(22:31), auch <https://youtu.be/ct81j3Y4z7I>

- Bearbeiten Sie das Quiz unter http://kleuker.iui.hs-osnabrueck.de/quiz/prog06_00236.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 3. 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 133f, 140ff

class Datum:

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

def get_tag(self) -> int:
    return self.tag

def set_tag(self, tag: int) -> None:
    self.tag = tag
```

```
def get_monat(self) -> int:
    return self.monat

def set_monat(self, monat: int) -> None:
    self.monat = monat

def get_jahr(self) -> int:
    return self.jahr

def set_jahr(self, jahr: int) -> None:
    self.jahr = jahr

class Datumsspielerei:
    def __init__(self) -> None:
        self.datum = Datum(29, 2, 2100)

    def einige_aenderungen(self) -> Datum:
        self.datum.set_tag(1)
        self.datum.set_monat(3)
        return self.datum

    def neuer_tag_neuer_monat(self, tag: int, monat: int) -> Datum:
        self.datum.set_tag(tag)
        self.datum.set_monat(monat)
        return self.datum

    def heiraten(self, wert: int) -> Datum:
        # genau wie in Java eine dumme Idee lokale Variable
        # heißt wie Objektvariable
        datum: Datum = Datum(wert, wert, wert)
        return datum

    def get_datum(self) -> Datum:
        return self.datum

d = Datum(20, 9, 2026)
print(f"Tag: {d.get_tag()}, Monat: {d.get_monat()}, Jahr: {d.get_jahr()}")
```

Angemerkt sei, dass get- und set-Methoden in Python eher selten genutzt werden. Dies liegt an der Historie der Sprache, da Objektorientierung erst spät ergänzt wurde. Wenn sich einmal unsaubere Techniken verbreiten, sind sie nicht mehr aufzuhalten. Wer sich Python-spezifisch damit beschäftigen möchte, sollte sich `@dataclass` ansehen. In Java zeigen Unterstriche (`_`) in Namen, dass die üblichen Java-Konventionen bei der programmierenden Person nicht bekannt sind. In Python ist das explizit anders, da finden Texttrennungen in Methodennamen üblicherweise mit Unterstrichen statt.

Folien 154ff

class Person:

```
def __init__(self) -> None:
    self.alter: int = 42
    self.name: str = "ich"
```

```
def naechstes(self) -> int:
    erg: int = self.alter + 1
    return erg

def merkwuerdig(self, wert: int) -> int:
    erg: int = len(self.name)
    erg = erg + wert
    return erg

def komisch(self, wert: int, w2: int) -> None:
    tmp: int = wert + w2
    self.alter = tmp

p: Person = Person()
v1: int = p.naechstes()
print(f"v1 ist {v1}")
v2: int = p.merkwuerdig(45)
print(f"v2 ist {v2}")
p.komisch(v1, v2)
print(f"p.alter ist {p.alter}, p.name ist {p.name}")

# v1 ist 43
# v2 ist 48
# p.alter ist 91, p.name ist ich
```

Folie 163

Die Klasse EinUndAusgabe.py liegt unter <https://kleuker.iui.hs-osnabrueck.de/querschnittlich/python/EinUndAusgabe.py> und muss in jedes Projekt kopiert werden, in dem sie genutzt wird.

Generell kann in Python, ähnlich wie in Java, jede Klasse in einer eigenen Datei stehen und in der nachfolgenden Form (from <Dateiname ohne .py> import <Klassenname>) geladen werden. In Python ist die Forderung „eine Klasse in einer Datei“ oft nicht umgesetzt, in Java wird es später mit dem Schlüsselwort „public“ notwendig (und sinnvoll). In Java müssen dabei Klassen, die im gleichen Verzeichnis liegen, nicht importiert werden. Dies macht die Notation in Python etwas aufwendiger, aber präziser, da bereits auf den ersten Blick deutlich wird, welche Klassen genutzt werden.

```
from EinUndAusgabe import EinUndAusgabe
```

```
class EinUndAusgabeAnalyse:
    def __init__(self) -> None:
        self.io: EinUndAusgabe = EinUndAusgabe()

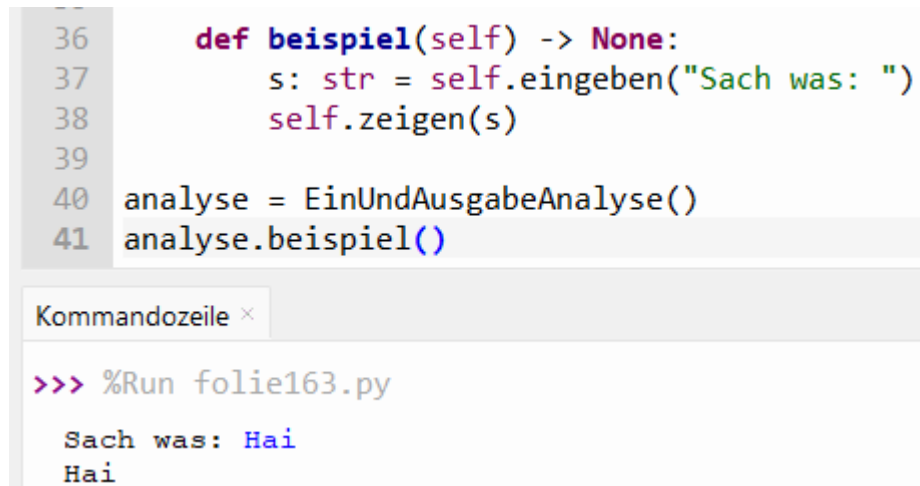
    def eingeben(self, befehl: str) -> str:
        self.io.ausgeben(befehl)
        ein: str = self.io lese_string()
        return ein

    def zeigen(self, text: str) -> None:
        self.io.ausgeben(text)
```

```
def beispiel(self) -> None:
    s: str = self.eingeben("Sach was: ")
    self.zeigen(s)

analyse = EinUndAusgabeAnalyse()
analyse.beispiel()
```

Die Ausführung sieht z. B. wie folgt aus.



```
36     def beispiel(self) -> None:
37         s: str = self.eingeben("Sach was: ")
38         self.zeigen(s)
39
40 analyse = EinUndAusgabeAnalyse()
41 analyse.beispiel()
```

Kommandozeile ×

```
>>> %Run folie163.py
Sach was: Hai
Hai
```

Annahme: Die Datei heißt EinUndAusgabe.py (oder ein_und_ausgabe.py) und liegt im gleichen Verzeichnis oder im PYTHONPATH.

Folie 165

```
from EinUndAusgabe import EinUndAusgabe
```

```
class Studierend:
    def __init__(
        self,
        vorname: str = "Eva",
        nachname: str = "Mustermann",
        geburtsjahr: int = 1990,
        studiengang: str = "IMI",
        matrikelnummer: int = 232323,
    ) -> None:
        self.vorname: str = vorname
        self.nachname: str = nachname
        self.geburtsjahr: int = geburtsjahr
        self.studiengang: str = studiengang
        self.matrikelnummer: int = matrikelnummer

    def get_vorname(self) -> str:
        return self.vorname

    def set_vorname(self, vorname: str) -> None:
        self.vorname = vorname

    def get_nachname(self) -> str:
        return self.nachname
```

```
def set_nachname(self, nachname: str) -> None:
    self.nachname = nachname

def get_geburtsjahr(self) -> int:
    return self.geburtsjahr

def set_geburtsjahr(self, geburtsjahr: int) -> None:
    self.geburtsjahr = geburtsjahr

def get_studiengang(self) -> str:
    return self.studiengang

def set_studiengang(self, studiengang: str) -> None:
    self.studiengang = studiengang

def get_matrikelnummer(self) -> int:
    return self.matrikelnummer

def set_matrikelnummer(self, matrikelnummer: int) -> None:
    self.matrikelnummer = matrikelnummer

def ausgeben(self) -> None:
    io: EinUndAusgabe = EinUndAusgabe()
    io.ausgeben("Vorname: ")
    io.ausgeben(self.vorname)
    io.ausgeben("\nNachname: ")
    io.ausgeben(self.nachname)
    io.ausgeben("\nStudiengang: ")
    io.ausgeben(self.studiengang)
    io.ausgeben("\n")
```

```
s1: Studierend = Studierend("Mr", "X", 1990, "IMI", 234243)
s2: Studierend = Studierend("Ms", "Y", 1980, "ITI", 234244)
s1.ausgeben()
s2.ausgeben()
s1 = s2
s2.set_vorname("King")
s1.ausgeben()
```

Ausgabe:
Vorname: Mr
Nachname: X
Studiengang: IMI
Vorname: Ms
Nachname: Y
Studiengang: ITI
Vorname: King
Nachname: Y
Studiengang: ITI

Folien 171ff

Unter <https://kleuker.iui.hs-osnabrueck.de/querschnittlich/python/Interaktionsbrett.py> kann eine Klasse Interaktionsbrett heruntergeladen werden, die eine Übertragbarkeit von Java-Ideen auf Python ermöglicht.

```
from EinUndAusgabe import EinUndAusgabe
from Interaktionsbrett import Interaktionsbrett

class Teilnehmend:
    def __init__(self, name: str, firma: str) -> None:
        self.name: str = name
        self.firma: str = firma

    def zeichnen(self, ib: Interaktionsbrett, x: int, y: int) -> None:
        ib.neuesRechteck(x, y, 100, 40)
        ib.neuerText(x + 2, y + 15, self.name)
        ib.neuerText(x + 2, y + 35, self.firma)

class Druckseite:
    def __init__(self) -> None:
        self.ib: Interaktionsbrett = Interaktionsbrett()
        self.y_position: int = 5

    def drucken(self, t: Teilnehmend) -> None:
        t.zeichnen(self.ib, 5, self.y_position)
        self.y_position = self.y_position + 50

    def schilder_erstellen(self) -> None:
        t: Teilnehmend = Teilnehmend("Carl Carlson", "Kraftwerk")
        self.drucken(t)

        t = Teilnehmend("Edna Krabbabel", "Grundschule")
        self.drucken(t)

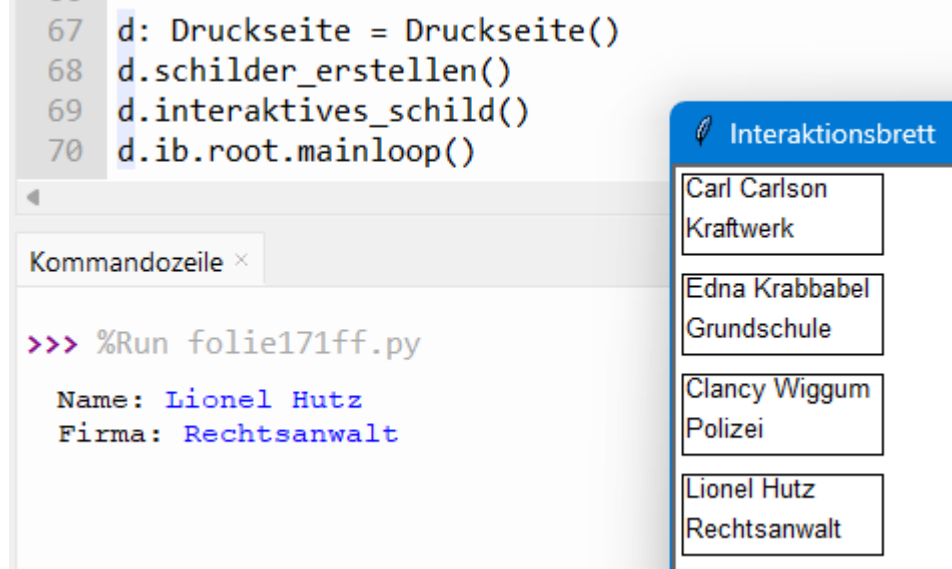
        self.drucken(Teilnehmend("Clancy Wiggum", "Polizei"))

    def interaktives_schild(self) -> None:
        io: EinUndAusgabe = EinUndAusgabe()
        io.ausgeben("Name: ")
        name: str = io.lese_string()
        io.ausgeben("Firma: ")
        unternehmen: str = io.lese_string()
        t: Teilnehmend = Teilnehmend(name, unternehmen)
        self.drucken(t)

d: Druckseite = Druckseite()
d.schilder_erstellen()
d.interaktives_schild()
# folgende Zeile benötigt, um das Interaktionsbrett anzuzeigen
# und damit es mit X auch schließbar ist
d.ib.root.mainloop()
```

Die Ausgabe sieht wie folgt aus.

```
67 d: Druckseite = Druckseite()
68 d.schilder_erstellen()
69 d.interaktives_schild()
70 d.ib.root.mainloop()
```



Kommandozeile ×

```
>>> %Run folie171ff.py

Name: Lionel Hutz
Firma: Rechtsanwalt
```

Interaktionsbrett

- Carl Carlson Kraftwerk
- Edna Krabbabel Grundschule
- Clancy Wiggum Polizei
- Lionel Hutz Rechtsanwalt

Wichtig ist der Kommentar kurz vor Ende des Programms, da so verhindert wird, dass das Interaktionsbrett vom Bildschirm verschwindet und dann auch danach geschlossen werden kann. Das Python-Interaktionsbrett verhält sich ähnlich, aber nicht identisch wie das Java-Interaktionsbrett, was auch diesen tiefen Eingriff in eine Objektvariable einer Objektvariable einer anderen Klasse zeigt. (Da der Quellcode vom Interaktionsbrett offen ist, bin ich für jede Verbesserung, bei Beibehaltung der gesamten Funktionalität dankbar.)

Folien 177ff

```
from EinUndAusgabe import EinUndAusgabe
from Interaktionsbrett import Interaktionsbrett
```

```
class Punkt:
    def __init__(self, x: int, y: int) -> None:
        self.x: int = x
        self.y: int = y

    def getX(self) -> int:
        return self.x

    def setX(self, x: int) -> None:
        self.x = x

    def getY(self) -> int:
        return self.y

    def setY(self, y: int) -> None:
        self.y = y

    def darstellen(self, ib: Interaktionsbrett) -> None:
        ib.neuerPunkt(self.x, self.y)

class Punktanalyse:
    def __init__(self) -> None:
```

```
self.ib: Interaktionsbrett = Interaktionsbrett()
self.io: EinUndAusgabe = EinUndAusgabe()

def punktEingeben(self) -> Punkt:
    self.io.ausgeben("X-Wert: ")
    x: int = self.io.lese_integer()
    self.io.ausgeben("Y-Wert: ")
    y: int = self.io.lese_integer()
    return Punkt(x, y)

def markieren(self, pkt: Punkt) -> None:
    self.ib.neuerPunkt(pkt.getX(), pkt.getY())
    self.ib.neuerKreis(pkt.getX() - 3, pkt.getY() - 3, 3)
    self.ib.neuerKreis(pkt.getX() - 6, pkt.getY() - 6, 6)

def benutzen(self) -> None:
    p1: Punkt = self.punktEingeben()
    self.markieren(p1)
    p2: Punkt = self.punktEingeben()
    self.markieren(p2)

analyse: Punktanalyse = Punktanalyse()
analyse.benutzen()
analyse.ib.root.mainloop()
```

Ein Beispielablauf sieht wie folgt aus.

