

Hinweis: Diese Lernnotiz enthält einen sehr sinnvollen Vorschlag um den Lehrstoff der 5. 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 21.10, 16:00 und Mo 2.11, 16:00, dazwischen liegt die Projektwoche; nutzen Sie diese zum Nacharbeiten) 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 9

- Syntax und Semantik der Alternative mit if-else verstehen und nutzen können
- Verständnis der Visualisierung der Alternative mit Aktivitätsdiagrammen und Fähigkeit zur eigenen Nutzung davon
- Verständnis von und Fähigkeit zur Nutzung geschachtelter Alternativen
- Boolesche Ausdrücke lesen, auswerten und schreiben können

Arbeitsschritte VL 9

- *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.*

Aktueller Stand, Einführung der Alternative

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Alternative1.mp4> (14:49),
auch https://youtu.be/LGb_XLolFYI

Beispielentwicklungen mit if in BlueJ

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Alternative2.mp4> (26:59),
auch <https://youtu.be/47-ZVzOhs7k>

Folien 227 – 242: if und Vergleichsoperationen

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Alternative3.mp4> (28:54),
auch <https://youtu.be/oLcvhvFJf8k>

Folien 243 – 249: Boolesche Ausdrücke

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Alternative4.mp4> (15:39),
auch <https://youtu.be/7YL3u5po6CA>

- Bearbeiten Sie das Quiz unter http://kleuker.iui.hs-osnabrueck.de/quiz/prog09_11237.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 10

- Fähigkeit Boolesche Ausdrücke umformen, deren semantische Gleichheit berechnen und die Kurzschussauswertung ausführen zu können
- Umgang mit int- und double-Ausdrücken mit den Werten Infinity, -Infinity und NaN kennen
- Sinn von Sichtbarkeiten kennen und nutzen können

Arbeitsschritte VL 10

- 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 250 – 252: Rechenregeln für Boolesche Ausdrücke

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Alternative5.mp4> (14:56),
auch <https://youtu.be/ePGzbVrjfrk>

Folien 253 – 267: Ausdrücke mit Zahlen, ?-Operator, Lebensspannen

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Alternative6.mp4> (38:47),
auch <https://youtu.be/hi5ielj5bkc>

Sichtbarkeiten und Umgang mit Null-Werten

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Alternative7.mp4> (26:36),
auch <https://youtu.be/88N5sYpEA4s>

- Bearbeiten Sie das Quiz unter http://kleuker.iui.hs-osnabrueck.de/quiz/prog10_11238.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.
- Bearbeiten Sie Aufgabenblatt 5. Denken Sie daran, dass ich für Fragen meist kurzfristig erreichbar bin.

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

Folie 227ff:

```
def persoenlichkeit(self) -> str:
    ergebnis: str
    if self.geburtsjahr < 1970:
        ergebnis = "alter Sack"
    else:
        ergebnis = "Windelhopser"
    return ergebnis

def genauerePersoenlichkeit(self) -> str:
    ergebnis: str
    if self.geburtsjahr < 1970:
        ergebnis = "alter Sack"
    else:
        if self.studiengang == "ITI":
            ergebnis = "Dynamit"
        else:
            ergebnis = "Windelhopser"
    return ergebnis
```

.equals() wird zu ==

```
def genauerePersoenlichkeit2(self) -> str:
    if self.geburtsjahr < 1970:
        return "alter Sack"
    if self.studiengang == "ITI":
        return "Dynamit"
    return "Windelhopser"

def genauerePersoenlichkeit3(self) -> str:
    ergebnis: str
    if self.geburtsjahr < 1970:
        ergebnis = "alter Sack"
    else:
        ergebnis = self.jungePersoenlichkeit()
    return ergebnis

def jungePersoenlichkeit(self) -> str:
    ergebnis: str
    if self.studiengang == "ITI":
        ergebnis = "Dynamit"
    else:
        ergebnis = "Windelhopser"
    return ergebnis

def langnameBerechnen(self) -> str:
    ergebnis: str = self.studiengang
    if self.studiengang == "ITI":
        ergebnis = "Informatik – Technische Informatik"
    return ergebnis
```

Generell ist die Ablauflogik identisch. Zu beachten ist, dass die inhaltliche Gleichheit in Python mit == überprüft wird. In Java wird == für identische Objekte, also eine Referenz auf das gleiche Objekt, genutzt. Die Identitätsüberprüfung in Python wird später betrachtet.

	Java	Python
Identität	==	is
Gleichheit	equals()	== (und später __eq__)

In Python wird oft auch folgende Variante genutzt, die in Java so nicht existiert:

```
def persoenlichkeit(self) -> str: return "alter Sack" \
    if self.geburtsjahr < 1970 else "Windelhopser"
```

Der Ausdruck hat allgemeiner die folgende Syntax:

<Wert> if <Boolesche Bedingung> else <Wert, wenn die Boolesche Bedingung nicht gilt>

Falls in Python ein Zeilenumbruch benötigt wird, kann dieser beliebig in Ausdrücken zwischen öffnenden und schließenden Klammern, rund oder eckig, stehen. Ist das, wie hier nicht möglich muss ein Backslash das letzte Zeichen der Zeile sein, nachfolgende Leerzeichen sind nicht erlaubt.

Folie 238

```
# Typisierte Übersetzung der Java-Zeilen aus der Abbildung
text1: str = "Hu" + "hu"
text3: str = "H" + "uhu"
```

```
print(text1 == text3)          # True
print(text1 is text3)          # True
print(type(text1 is text3))    # <class 'bool'>

s1: str = text1.replace("u", "a")
s2: str = text3.replace("u", "a")

print(s1)                      # Haha
print(type(s1))                # <class 'str'>

print(s2)                      # Haha
print(type(s2))                # <class 'str'>

# Java: s1 == s2 ist false (verschiedene Objekte)
print(s1 is s2)                # False

# Java: s1.equals(s2) ist true (gleicher Inhalt)
print(s1 == s2)                # True
```

Das Beispiel zeigt, dass == und equals() aus Java in Python anders genutzt werden. Neben == gibt es in Python is, was vereinfacht dem == in Java entspricht.

Folie 246

```
y: int = 500
z: int = 500
print(y == z)                  # True
print(y != z)                  # False
print(type(y != z))           # <class 'bool'>
```

Folie 249

```
alter: int = 23
iq: int = 42

print(alter > iq)               # False
# (and hat höhere Priorität als or, wie in Java auch)
print(alter > 20 or iq < 10 and iq > 99) # True
print((alter > 20 or iq < 10) and iq > 99) # False
print(not (alter > iq))         # False
```

Zu beachten ist, dass in Python die Booleschen Operatoren ausgeschrieben werden.

Folie 254

```
result1: int = 7 + 4 * 3
print(result1)                  # 19
print(type(result1))           # <class 'int'>

result2: int = (7 + 4) * 3
print(result2)                  # 33

result3: int = 7 / 4            # nicht wie in Java
print(result3)                  # 1.75

result3b: int = 7 // 4          # // Floor-Division (entspricht Java-/)
print(result3b)                 # 1

result4: int = 7 % 4
print(result4)                  # 3
```

```
print(0 % 2)          # 0
print(7 + 4 > 5 + 6)   # False
print(7 % (1 - 7 / 4)) # -0.5
print(7 % (1 - 7 // 4)) # ZeroDivisionError: integer modulo by zero
```

Python nutzt leider nicht den IEEE 754-Standard für mathematische Operationen, was sich z. B. bei der ganzzahligen Division zeigt, die bei int-Werten nicht mit / sondern mit // stattfindet.

Folie 256

```
result1: float = 7.3 + 5.4 * 3.8
print(result1)          # 27.82
print(type(result1))    # <class 'float'>

result2: float = (7.3 + 5.4) * 3.8
print(result2)          # 48.26

result3: bool = 7.3 > 5.3 + 2.4 or 5.5 + 4.4 < 3.3 + 6.6
print(result3)          # False

# result4: 7.3 / 0
# In Python wird ZeroDivisionError ausgelöst.
result4 = float("inf")   # Infinity in Java
print(result4)           # inf
print(type(result4))     # <class 'float'>

result5 = float("-inf")  # -Infinity in Java
print(result5)           # -inf
print(type(result5))     # <class 'float'>

result6 = float("nan")   # NaN in Java
print(result6)           # nan
print(type(result6))     # <class 'float'>
print(result4 - result5) # inf
print(result4 + result5) # nan
print(result4 / result4) # nan
print(result4 * 0)       # nan
print(result6 * 0)       # nan
print(result4 + 1)       # inf
print(result4 - 1)       # inf
print(7.0 % 3.4)         # 0.200000000000000018
```

In Python führt jedwede Division durch die Zahl Null anders als in Java und im Standard zu einer Fehlermeldung. Die für Berechnungen wichtigen Werte Infinity, -Infinity und NaN aus Java sind als inf, -inf und nan trotzdem vorhanden und verhalten sich wie in Java und IEEE 754. Der Typ float aus Python hat die Rechengenauigkeit von double in Java.

Folie 257ff:

```
def langnameBerechnen2(self) -> str:
    ergebnis: str = self.studiengang
    if self.studiengang == "ITI":
        return "Informatik - Technische Informatik"
    return ergebnis

def langnameBerechnen3(self) -> str:
    if self.studiengang == "ITI":
        return "Informatik - Technische Informatik"
    return self.studiengang
```

```
def langnameBerechnen_ternary(self) -> str:
    # Java-Ternary-Operator ? : hat andere vorher erwähnte Syntax
    return ("Informatik – Technische Informatik"
            if self.studiengang == "ITI"
            else self.studiengang)
```

Die letzte Methode zeigt, wie ein explizites Symbol für den Zeilenumbruch durch eigentlich unnötige runde Klammern vermieden wird, da in Klammern an beliebigen Stellen umgebrochen werden kann.

Folie 263f

Das Problem mit dem falsch platzierten else ist in Python auch möglich, aber unwahrscheinlicher, da die Einrückung angibt, zu welchen if das else gehört.

class Verwaltung:

```
def istLoginKorrekt(self, nutzend: str, password: str) -> bool:
    return nutzend == "admin" and password == "geheim"

def alsGastErlaubt(self, nutzend: str, password: str) -> bool:
    return nutzend == "gast"

def gruppe(self, nutzend: str, password: str) -> str:
    ergebnis: str = "keine Rechte"
    if not self.istLoginKorrekt(nutzend, password):
        if self.alsGastErlaubt(nutzend, password):
            ergebnis = "Gast"
        else:
            ergebnis = "Admin"
    return ergebnis

def gruppeOK(self, nutzend: str, password: str) -> str:
    """Korrekt geklammerte Version."""
    ergebnis: str = "keine Rechte"
    if not self.istLoginKorrekt(nutzend, password):
        if self.alsGastErlaubt(nutzend, password):
            ergebnis = "Gast"
    else:
        ergebnis = "Admin"
    return ergebnis
```

s = Verwaltung()

```
print("\n--- gruppe (dangling else) ---")
print("admin/geheim:", s.gruppe("admin", "geheim"))
print("gast/xyz      :", s.gruppe("gast", "xyz"))
print("user/xyz      :", s.gruppe("user", "xyz"))

print("\n--- gruppeOK (korrekt) ---")
print("admin/geheim:", s.gruppeOK("admin", "geheim"))
print("gast/xyz      :", s.gruppeOK("gast", "xyz"))
print("user/xyz      :", s.gruppeOK("user", "xyz"))
```

Die zugehörige Ausgabe lautet:

```
--- gruppe (dangling else) ---
admin/geheim: keine Rechte
gast/xyz      : Gast
user/xyz      : Admin
```

```
--- gruppeOK (korrekt) ---  
admin/geheim: Admin  
gast/xyz      : Gast  
user/xyz      : keine Rechte
```

Python hat allerdings ein Einrückungsproblem, was gerade bei Programmänderungen übersehen werden kann. Das sind fehlende Einrückungen. Ein Beispiel zeigt das folgende Programm:

```
class Punkt:  
    def __init__(self, x: int, y: int) -> None:  
        self.__x: int = x  
        self.__y: int = y  
  
    def nach_links_oben(self) -> None:  
        if self.__x > 0 and self.__y > 0:  
            self.__x = self.__x - 1  
            self.__y = self.__y - 1  
  
    def nach_links_obenOK(self) -> None:  
        if self.__x > 0 and self.__y > 0:  
            self.__x = self.__x - 1  
            self.__y = self.__y - 1  
  
    def ausgeben(self) -> None:  
        print(f"({self.__x},{self.__y})")  
  
p1: Punkt = Punkt (42, 161)  
p1.nach_links_oben()  
p1.ausgeben()  
p2: Punkt = Punkt (42, 161)  
p2.nach_links_obenOK()  
p2.ausgeben()  
p3: Punkt = Punkt (42, 0)  
p3.nach_links_oben()  
p3.ausgeben()  
p4: Punkt = Punkt (42, 0)  
p4.nach_links_obenOK()  
p4.ausgeben()
```

Die Ausgabe lautet:

```
(41,160)  
(41,160)  
(42,-1)  
(42,0)
```

Folie 266

Da Objektorientierung erst später in Python eingearbeitet wurde, gibt es mehrere Kompromisse zur sauberen Objektorientierung. Ein wesentlicher ist der Workaround, um Sichtbarkeiten einzuführen. Grundsätzlich kann in Python auf beliebig weit innen liegende Variablen zugegriffen werden. Um dies zu erschweren und entwickelnden Personen deutlich zu machen, dass es sich um `private`-Elemente handelt, werden dann dem jeweiligen Namen zwei Unterstriche(`__`) vorangestellt. Unterstriche nach dem Namen sind zu vermeiden, da diese eine andere Bedeutung haben, die später betrachtet wird. Das folgende Beispiel zeigt aber, dass dies durch eine Namensergänzung umgangen werden kann.

```
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:
        # private Attribute (Java: private)
        self.__vorname: str = vorname
        self.__nachname: str = nachname
        self.__geburtsjahr: int = geburtsjahr
        self.__studiengang: str = studiengang
        self.__matrikelnummer: int = matrikelnummer

    def getVorname(self) -> str:
        return self.__vorname

    ...
    def __jungePersoenlichkeit(self) -> str:
        ergebnis: str
        if self.__studiengang == "ITI":
            ergebnis = "Dynamit"
        else:
            ergebnis = "Windelhopser"
        return ergebnis

print("-- Sichtbarkeiten austricksen -- ")
s: Studierend = Studierend()
print(s._Studierend__vorname)
print(s._Studierend__jungePersoenlichkeit())
```

Die Ausgabe lautet:

```
-- Sichtbarkeiten austricksen --
Eva
Windelhopser
```

Folie 267:

Die Ideen aus Java können teilweise übernommen werden. Einmal deklarierte Variablen sind auch später nutzbar. Im folgenden Beispiel ist dies an `lokalVar2` zu erkennen, die im `if`-Zweig deklariert wird und so in der gesamten Methode nutzbar ist. Dies ist typisch für Skriptsprachen und untypisch für fast alle anderen Sprachen, wie C, C++, C#, Go, Rust.

```
class Klasse:

    def __init__(self) -> None:
        self.__objektVar: int = 0 # private int objektVar;

    def methode1(self) -> None:
        lokalVar: int = 41
        # ...
        self.methode2(lokalVar)
```

```
def methode2(self, param: int) -> None:
    lokalVar: int    # geht in Python, erzeugt aber keine noch keine Variable
    # ...
    if param == 42:
        lokalVar2: int    # in Java nur innerhalb des if-Blocks sichtbar
        lokalVar2 = 99
        # ...
        print(f"lokalVar2 = {lokalVar2}") # in Java nur hier gültig
    print("lokalVar2: ")
    print(lokalVar2)      # in Python auch denkbar

k = Klasse()
k.methode2(42)
k.methode1()
```

Die Ausgabe lautet:

```
lokalVar2 = 99
lokalVar2:
99
lokalVar2:
Traceback (most recent call last):
  File "<string>", line 26, in <module>
  File "<string>", line 10, in methode1
  File "<string>", line 21, in methode2
UnboundLocalError: cannot access local variable 'lokalVar2' where it is not
associated with a value
```

Beim ersten Aufruf mit `k.methode2(42)` wird der if-Zweig durchlaufen und die Variable `lokalVar2` deklariert. Sie ist dann auch außerhalb des if-Zweigs nutzbar. Am Ende der Methode sterben, wie in Java, alle lokalen Variablen. Beim Aufruf `k.methode1()` wird der if-Zweig nicht durchlaufen, also wird die Variable nicht deklariert, ist damit außerhalb nicht nutzbar und führt zu einem Fehler. Dies macht deutlich, dass das „Variablenleben“ in Python etwas flexibler ist, dies aber in der Standardprogrammierung nicht genutzt werden sollte.