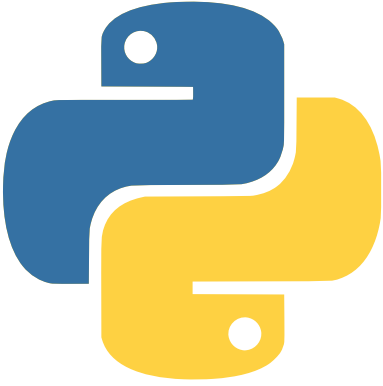
**Universität Stuttgart**
Projekt digit@L – BOOST. SKILLS. SUPPORT.



# Programmierkurs Python

## Variablen und Zuweisungen

Dominik Göddeke

IANS SimTech

Stiftung Innovation in der Hochschullehre

# Variablen und Zuweisungen

# Variablen und Zuweisungen

- Im abstrakten Computermodell: Daten liegen irgendwo im Speicher

# Variablen und Zuweisungen

- Im abstrakten Computermodell: Daten liegen irgendwo im Speicher
- Speicheradressen nicht Menschen-lesbar, und stark Hardware-abhängig

# Variablen und Zuweisungen

- Im abstrakten Computermodell: Daten liegen irgendwo im Speicher
- Speicheradressen nicht Menschen-lesbar, und stark Hardware-abhängig
  - Unterschiedliche Adressen bei verschiedenen Läufen desselben Programms

# Variablen und Zuweisungen

- Im abstrakten Computermodell: Daten liegen irgendwo im Speicher
- Speicheradressen nicht Menschen-lesbar, und stark Hardware-abhängig
  - Unterschiedliche Adressen bei verschiedenen Läufen desselben Programms
- Deshalb: **Variablen** als lesbares Synonym einer Speicheradresse

# Variablen und Zuweisungen

- Im abstrakten Computermodell: Daten liegen irgendwo im Speicher
- Speicheradressen nicht Menschen-lesbar, und stark Hardware-abhängig
  - Unterschiedliche Adressen bei verschiedenen Läufen desselben Programms
- Deshalb: **Variablen** als lesbares Synonym einer Speicheradresse
- Definition über **Zuweisung**

```
variablenname = wert
```

# Variablen und Zuweisungen

- Im abstrakten Computermodell: Daten liegen irgendwo im Speicher
- Speicheradressen nicht Menschen-lesbar, und stark Hardware-abhängig
  - Unterschiedliche Adressen bei verschiedenen Läufen desselben Programms
- Deshalb: **Variablen** als lesbares Synonym einer Speicheradresse
- Definition über **Zuweisung**

```
variablenname = wert
```

- Python: sogenannte **dynamisch typisierte Sprache**

# Variablen und Zuweisungen

- Im abstrakten Computermodell: Daten liegen irgendwo im Speicher
- Speicheradressen nicht Menschen-lesbar, und stark Hardware-abhängig
  - Unterschiedliche Adressen bei verschiedenen Läufen desselben Programms
- Deshalb: **Variablen** als lesbares Synonym einer Speicheradresse
- Definition über **Zuweisung**

```
variablenname = wert
```

- Python: sogenannte **dynamisch typisierte Sprache**
  - **Datentyp** der Variable (wie Zeichenkette, Zahl, irgendein komplexes Objekt) nicht explizit angegeben

# Variablen und Zuweisungen

- Im abstrakten Computermodell: Daten liegen irgendwo im Speicher
- Speicheradressen nicht Menschen-lesbar, und stark Hardware-abhängig
  - Unterschiedliche Adressen bei verschiedenen Läufen desselben Programms
- Deshalb: **Variablen** als lesbares Synonym einer Speicheradresse
- Definition über **Zuweisung**

```
variablenname = wert
```

- Python: sogenannte **dynamisch typisierte Sprache**
  - **Datentyp** der Variable (wie Zeichenkette, Zahl, irgendein komplexes Objekt) nicht explizit angegeben
  - Sondern implizit durch den Typ der rechten Seite der Zuweisung

# Ein paar Beispiele . . .

# Variablen und Zuweisungen im Detail

# Variablen und Zuweisungen im Detail

- **Variablenzuweisung** ist **keine Gleichung** im mathematischen Sinne, trotz des Gleichheitszeichens

# Variablen und Zuweisungen im Detail

- **Variablenzuweisung** ist **keine Gleichung** im mathematischen Sinne, trotz des Gleichheitszeichens
- Keine Definition der Variable durch sich selbst

```
z = 2 - z
```

# Variablen und Zuweisungen im Detail

- **Variablenzuweisung** ist **keine Gleichung** im mathematischen Sinne, trotz des Gleichheitszeichens
- Keine Definition der Variable durch sich selbst

```
z = 2 - z
```

- Folgendes ist hingegen **erlaubt**

```
a = 1; b = a + 2; print(a,b)
```

# Variablen und Zuweisungen im Detail

- **Daumenregel** für das Verständnis

# Variablen und Zuweisungen im Detail

- **Daumenregel** für das Verständnis
  - **Zuweisung füllt/belegt** die Variable „links vom Zuweisungszeichen =“

# Variablen und Zuweisungen im Detail

- **Daumenregel** für das Verständnis
  - **Zuweisung füllt/belegt** die Variable „links vom Zuweisungszeichen =“
  - Und zwar **mit Wert** rechts vom Zuweisungszeichen, der sich durch die Werte der Variablen rechts und deren Verbindung durch Operationen ergibt

# Variablen und Zuweisungen im Detail

- **Daumenregel** für das Verständnis
  - **Zuweisung füllt/belegt** die Variable „links vom Zuweisungszeichen =“
  - Und zwar **mit Wert** rechts vom Zuweisungszeichen, der sich durch die Werte der Variablen rechts und deren Verbindung durch Operationen ergibt
  - Linke Seite hat den Typ der rechten Seite

# Variablen und Zuweisungen im Detail

- **Daumenregel** für das Verständnis
  - **Zuweisung füllt/belegt** die Variable „links vom Zuweisungszeichen =“
  - Und zwar **mit Wert** rechts vom Zuweisungszeichen, der sich durch die Werte der Variablen rechts und deren Verbindung durch Operationen ergibt
  - Linke Seite hat den Typ der rechten Seite
- **Quiz**: Warum ist das Folgende erlaubt?

```
z = 5; z = 2 - z; print(z)
```

  - **Tipp**: Probieren Sie die Einzeiler selbst in Python aus, bevor Sie antworten

# Zulässige und unzulässige Namen von Variablen

- **Erlaubte Zeichen** für Variablennamen

# Zulässige und unzulässige Namen von Variablen

- **Erlaubte Zeichen** für Variablennamen
  - Großbuchstaben `A` bis `Z`, Kleinbuchstaben `a` bis `z`

# Zulässige und unzulässige Namen von Variablen

- **Erlaubte Zeichen** für Variablennamen
  - Großbuchstaben `A` bis `Z`, Kleinbuchstaben `a` bis `z`
  - Unterstrich _

# Zulässige und unzulässige Namen von Variablen

- **Erlaubte Zeichen** für Variablennamen
  - Großbuchstaben `A` bis `Z`, Kleinbuchstaben `a` bis `z`
  - Unterstrich _
  - Ziffern `0` bis `9` überall **außer an erster Stelle**

# Zulässige und unzulässige Namen von Variablen

- **Erlaubte Zeichen** für Variablennamen
  - Großbuchstaben `A` bis `Z`, Kleinbuchstaben `a` bis `z`
  - Unterstrich _
  - Ziffern `0` bis `9` überall **außer an erster Stelle**
- **Unzulässige Variablennamen**

```
False     class      finally    is         return
None      continue   for        lambda     try
True      def        from       nonlocal   while
and       del        global     not        with
as        elif       if         or         yield
assert    else       import     pass
break     except     in         raise
```

# Zulässige und unzulässige Namen von Variablen

- **Erlaubte Zeichen** für Variablennamen
  - Großbuchstaben `A` bis `Z`, Kleinbuchstaben `a` bis `z`
  - Unterstrich _
  - Ziffern `0` bis `9` überall **außer an erster Stelle**
- **Unzulässige Variablennamen**

```
False     class      finally    is         return
None      continue   for        lambda     try
True      def        from       nonlocal   while
and       del        global     not        with
as        elif       if         or         yield
assert    else       import     pass
break     except     in         raise
```

- Spezielle Bedeutung dieser Bezeichner, mehr im Kursverlauf

# Zulässige und unzulässige Namen von Variablen

- **Erlaubte Zeichen** für Variablennamen
  - Großbuchstaben `A` bis `Z`, Kleinbuchstaben `a` bis `z`
  - Unterstrich `_`
  - Ziffern `0` bis `9` überall **außer an erster Stelle**
- **Unzulässige Variablennamen**

```
False    class      finally    is         return
None     continue   for        lambda     try
True     def        from       nonlocal   while
and      del        global     not        with
as       elif       if         or         yield
assert   else       import     pass
break    except     in         raise
```

- Spezielle Bedeutung dieser Bezeichner, mehr im Kursverlauf
- Achtung: Python unterscheidet zwischen **Groß- und Kleinschreibung**

# Sinnvolle Konventionen zur Benennung von Variablen

# Sinnvolle Konventionen zur Benennung von Variablen

- **Goldene Regel**: einheitliches Benennungsschema nach etablierten Konventionen („coding conventions“)

# Sinnvolle Konventionen zur Benennung von Variablen

- **Goldene Regel**: einheitliches Benennungsschema nach etablierten Konventionen („coding conventions“)
  - Kleinbuchstaben mit Unterstrichen: `variable_one = 1`

# Sinnvolle Konventionen zur Benennung von Variablen

- **Goldene Regel**: einheitliches Benennungsschema nach etablierten Konventionen („coding conventions“)
  - Kleinbuchstaben mit Unterstrichen: `variable_one = 1`
  - CamelCase 1: Großbuchstaben statt Unterstriche: `variableOne = 1`

# Sinnvolle Konventionen zur Benennung von Variablen

- **Goldene Regel**: einheitliches Benennungsschema nach etablierten Konventionen („coding conventions“)
  - Kleinbuchstaben mit Unterstrichen: `variable_one = 1`
  - CamelCase 1: Großbuchstaben statt Unterstriche: `variableOne = 1`
  - CamelCase 2: konsequente Verwendung von Großbuchstaben: `VariableOne=1`

# Sinnvolle Konventionen zur Benennung von Variablen

- **Goldene Regel**: einheitliches Benennungsschema nach etablierten Konventionen („coding conventions“)
  - Kleinbuchstaben mit Unterstrichen: `variable_one = 1`
  - CamelCase 1: Großbuchstaben statt Unterstriche: `variableOne = 1`
  - CamelCase 2: konsequente Verwendung von Großbuchstaben: `VariableOne=1`
- **Jede Variante zulässig**, ebenso weitere Schemata

# Sinnvolle Konventionen zur Benennung von Variablen

- **Goldene Regel**: einheitliches Benennungsschema nach etablierten Konventionen („coding conventions“)
  - Kleinbuchstaben mit Unterstrichen: `variable_one = 1`
  - CamelCase 1: Großbuchstaben statt Unterstriche: `variableOne = 1`
  - CamelCase 2: konsequente Verwendung von Großbuchstaben: `VariableOne=1`
- **Jede Variante zulässig**, ebenso weitere Schemata
- Gute (gut lesbare) Programme verwenden ein **einheitliches Schema**

# Sinnvolle Konventionen zur Benennung von Variablen

- **Goldene Regel**: einheitliches Benennungsschema nach etablierten Konventionen („coding conventions“)
  - Kleinbuchstaben mit Unterstrichen: `variable_one = 1`
  - CamelCase 1: Großbuchstaben statt Unterstriche: `variableOne = 1`
  - CamelCase 2: konsequente Verwendung von Großbuchstaben: `VariableOne=1`
- **Jede Variante zulässig**, ebenso weitere Schemata
- Gute (gut lesbare) Programme verwenden ein **einheitliches Schema**
- Fun fact: epischer Meinungsstreit in der Community, `https://www.python.org/dev/peps/pep-0008`

# Sinnvolle Konventionen zur Benennung von Variablen

- Schreibweise von Variablen in diesem Kurs

# Sinnvolle Konventionen zur Benennung von Variablen

- Schreibweise von Variablen in diesem Kurs
  - **Kleinbuchstaben mit Unterstrichen** wegen pep-0008

# Sinnvolle Konventionen zur Benennung von Variablen

- Schreibweise von Variablen in diesem Kurs
  - **Kleinbuchstaben mit Unterstrichen** wegen pep-0008
  - **Englische Bezeichner für Variablen**

# Sinnvolle Konventionen zur Benennung von Variablen

- Schreibweise von Variablen in diesem Kurs
  - **Kleinbuchstaben mit Unterstrichen** wegen pep-0008
  - **Englische Bezeichner für Variablen**
  - Als Konsequenz: englische Sprache in Kommentaren

# Sinnvolle Konventionen zur Benennung von Variablen

- Schreibweise von Variablen in diesem Kurs
  - **Kleinbuchstaben mit Unterstrichen** wegen pep-0008
  - **Englische Bezeichner für Variablen**
  - Als Konsequenz: englische Sprache in Kommentaren
- Ebenso wichtig: **Name einer Variable spiegelt den Inhalt wider**

# Sinnvolle Konventionen zur Benennung von Variablen

- Schreibweise von Variablen in diesem Kurs
  - **Kleinbuchstaben mit Unterstrichen** wegen pep-0008
  - **Englische Bezeichner für Variablen**
  - Als Konsequenz: englische Sprache in Kommentaren
- Ebenso wichtig: **Name einer Variable spiegelt den Inhalt wider**
- Erspart viele Kommentare

# Einige Code-Beispiele

# Interner Umgang mit Variablen

# Interner Umgang mit Variablen

- Ziel: **Verständnis der Details der Speicher-Abstraktion** durch Experimente

# Interner Umgang mit Variablen

- Ziel: **Verständnis der Details der Speicher-Abstraktion** durch Experimente
- Hochrelevant für uns (!)

# Interner Umgang mit Variablen

- Ziel: **Verständnis der Details der Speicher-Abstraktion** durch Experimente
- Hochrelevant für uns (!)
- Ein kleiner Codeschnipsel:

```
var1 = 1; var2 = var1
print("Vorher: ", var1, var2")
var1 = 4
print("Nachher: ", var1, var2")
```

# Interner Umgang mit Variablen

- Ziel: **Verständnis der Details der Speicher-Abstraktion** durch Experimente
- Hochrelevant für uns (!)
- Ein kleiner Codeschnipsel:

```
var1 = 1; var2 = var1
print("Vorher: ", var1, var2")
var1 = 4
print("Nachher: ", var1, var2")
```

- Ausgabe: `Vorher: 1 1`, `Nachher: 4 1`

# Interner Umgang mit Variablen

```
var1 = 1; var2 = var1
print("Vorher: ", var1, var2")
var1 = 4
print("Nachher: ", var1, var2")
```

- Im Hintergrund, **bisheriges Verständnis**

# Interner Umgang mit Variablen

```
var1 = 1; var2 = var1
print("Vorher: ", var1, var2")
var1 = 4
print("Nachher: ", var1, var2")
```

- Im Hintergrund, **bisheriges Verständnis**
  - `var2 = var1`: unklar, zwei Namen für dieselbe Speicheradresse oder Kopie in eine neue Speicheradresse?

# Interner Umgang mit Variablen

```
var1 = 1; var2 = var1
print("Vorher: ", var1, var2")
var1 = 4
print("Nachher: ", var1, var2")
```

- Im Hintergrund, **bisheriges Verständnis**
  - `var2 = var1`: unklar, zwei Namen für dieselbe Speicheradresse oder Kopie in eine neue Speicheradresse?
  - `var1 = 4` überschreiben an der Speicheradresse?

# Interner Umgang mit Variablen

```
var1 = 1; var2 = var1
print("Vorher: ", var1, var2")
var1 = 4
print("Nachher: ", var1, var2")
```

- Im Hintergrund, **bisheriges Verständnis**
  - `var2 = var1`: unklar, zwei Namen für dieselbe Speicheradresse oder Kopie in eine neue Speicheradresse?
  - `var1 = 4` überschreiben an der Speicheradresse?
- **Besseres Verständnis**: `id()` Funktion

# Interner Umgang mit Variablen

```
var1 = 1; var2 = var1
print("Vorher: ", var1, var2")
var1 = 4
print("Nachher: ", var1, var2")
```

- Im Hintergrund, **bisheriges Verständnis**
  - `var2 = var1`: unklar, zwei Namen für dieselbe Speicheradresse oder Kopie in eine neue Speicheradresse?
  - `var1 = 4` überschreiben an der Speicheradresse?
- **Besseres Verständnis**: `id()` Funktion
  - Liefert die interne ID zurück, die Python zur Identifikation verwendet

# Interner Umgang mit Variablen

```
var1 = 1; var2 = var1
print("Vorher: ", var1, var2")
var1 = 4
print("Nachher: ", var1, var2")
```

- Im Hintergrund, **bisheriges Verständnis**
  - `var2 = var1`: unklar, zwei Namen für dieselbe Speicheradresse oder Kopie in eine neue Speicheradresse?
  - `var1 = 4` überschreiben an der Speicheradresse?
- **Besseres Verständnis**: `id()` Funktion
  - Liefert die interne ID zurück, die Python zur Identifikation verwendet
  - Gewissermaßen eine abstrakte Hardware-unabhängige „Speicheradresse“ innerhalb der aktuellen Ausführung genau dieses Python-Programms

# Wir untersuchen das im Programm . . .

# Interner Umgang mit Variablen

- Zuweisungsoperation `var2=var1` lässt beide Variablen-Namen auf dasselbe „Objekt“ im Speicher „zeigen“

# Interner Umgang mit Variablen

- Zuweisungsoperation `var2=var1` lässt beide Variablen-Namen auf dasselbe „Objekt“ im Speicher „zeigen“
- Ausgabe der Werte der Variablen erwartbar identisch – **probieren Sie es aus**

# Interner Umgang mit Variablen

- Zuweisungsoperation `var2=var1` lässt beide Variablen-Namen auf dasselbe „Objekt“ im Speicher „zeigen“
- Ausgabe der Werte der Variablen erwartbar identisch – **probieren Sie es aus**
- Zuweisung erzeugt einen **Alias** / einen synonymen Namen für die Variable

# Interner Umgang mit Variablen

- Zuweisungsoperation `var2=var1` lässt beide Variablen-Namen auf dasselbe „Objekt“ im Speicher „zeigen“
- Ausgabe der Werte der Variablen erwartbar identisch – **probieren Sie es aus**
- Zuweisung erzeugt einen **Alias** / einen synonymen Namen für die Variable
- Aber (zunächst) **keine Kopie**

# Interner Umgang mit Variablen

- Zuweisungsoperation `var2=var1` lässt beide Variablen-Namen auf dasselbe „Objekt“ im Speicher „zeigen“
- Ausgabe der Werte der Variablen erwartbar identisch – **probieren Sie es aus**
- Zuweisung erzeugt einen **Alias** / einen synonymen Namen für die Variable
- Aber (zunächst) **keine Kopie**
- Sprechweise: **by-reference** Kopie

# Interner Umgang mit Variablen

- Zuweisungsoperation `var2=var1` lässt beide Variablen-Namen auf dasselbe „Objekt“ im Speicher „zeigen“
- Ausgabe der Werte der Variablen erwartbar identisch – **probieren Sie es aus**
- Zuweisung erzeugt einen **Alias** / einen synonymen Namen für die Variable
- Aber (zunächst) **keine Kopie**
- Sprechweise: **by-reference** Kopie
  - Sehr sinnvoll, „referenzierter Datensatz“ kann durchaus riesig sein, und nicht nur eine Zahl wie im Beispiel

# Interner Umgang mit Variablen

- Zuweisungsoperation `var2=var1` lässt beide Variablen-Namen auf dasselbe „Objekt“ im Speicher „zeigen“
- Ausgabe der Werte der Variablen erwartbar identisch – **probieren Sie es aus**
- Zuweisung erzeugt einen **Alias** / einen synonymen Namen für die Variable
- Aber (zunächst) **keine Kopie**
- Sprechweise: **by-reference** Kopie
  - Sehr sinnvoll, „referenzierter Datensatz“ kann durchaus riesig sein, und nicht nur eine Zahl wie im Beispiel
  - Bspw. mehrere TeraByte Daten für das Training eines Machine-Learning Modells

# Interner Umgang mit Variablen

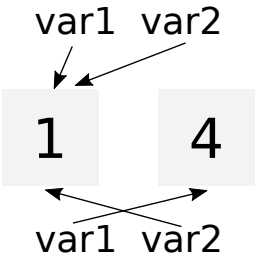
- Zuweisungsoperation `var2=var1` lässt beide Variablen-Namen auf dasselbe „Objekt“ im Speicher „zeigen“
- Ausgabe der Werte der Variablen erwartbar identisch – **probieren Sie es aus**
- Zuweisung erzeugt einen **Alias** / einen synonymen Namen für die Variable
- Aber (zunächst) **keine Kopie**
- Sprechweise: **by-reference** Kopie
  - Sehr sinnvoll, „referenzierter Datensatz“ kann durchaus riesig sein, und nicht nur eine Zahl wie im Beispiel
  - Bspw. mehrere TeraByte Daten für das Training eines Machine-Learning Modells
  - Implizite Kosten einer (unbeabsichtigten) Kopie potentiell enorm, Kopie einer **Referenz** aber quasi umsonst

# Interner Umgang mit Variablen

- Zuweisungsoperation `var2=var1` lässt beide Variablen-Namen auf dasselbe „Objekt“ im Speicher „zeigen“
- Ausgabe der Werte der Variablen erwartbar identisch – **probieren Sie es aus**
- Zuweisung erzeugt einen **Alias** / einen synonymen Namen für die Variable
- Aber (zunächst) **keine Kopie**
- Sprechweise: **by-reference** Kopie
  - Sehr sinnvoll, „referenzierter Datensatz“ kann durchaus riesig sein, und nicht nur eine Zahl wie im Beispiel
  - Bspw. mehrere TeraByte Daten für das Training eines Machine-Learning Modells
  - Implizite Kosten einer (unbeabsichtigten) Kopie potentiell enorm, Kopie einer **Referenz** aber quasi umsonst
- Konsequenz: automatisches Löschen nicht mehr referenzierter Speicherstellen

# Interner Umgang mit Variablen

```
var1 = 1; var2 = var1
print("Vorher: ", var1, var2")
var1 = 4
print("Nachher: ", var1, var2")
```



# Mehrfache Variablenzuweisung

# Mehrfache Variablenzuweisung

- Mehrere Variablen können gleichzeitig Werte zugewiesen bekommen

# Mehrfache Variablenzuweisung

- Mehrere Variablen können gleichzeitig Werte zugewiesen bekommen
- Intern: sogenannte **Tupel**

# Mehrfache Variablenzuweisung

- Mehrere Variablen können gleichzeitig Werte zugewiesen bekommen
- Intern: sogenannte **Tupel**
- Verschiedene Daten in fester Reihenfolge

# Schauen wir es uns an . . .

# Impressum, Danksagung und Quellen

Stiftung Innovation in der Hochschullehre

Gefördert durch die Stiftung Innovation in der Hochschullehre im Rahmen des Projekts digit@L, https://stiftung-hochschullehre.de

SimTech

Gefördert mit Mitteln der Deutschen Forschungsgemeinschaft (EXC 2075 - 390740016) im Rahmen der Exzellenzstrategie

---

Autor: Dominik Göddeke, IANS, Universität Stuttgart



---

ZOERR

Veröffentlicht auf dem Zentralen OER Repositorium Baden-Württemberg, https://www.zoerr.de