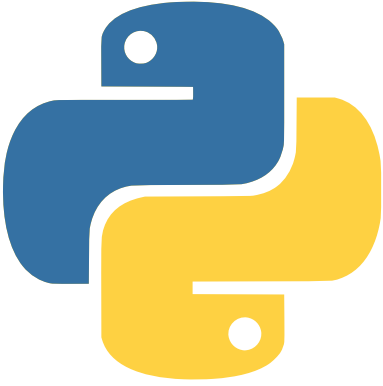
Universität Stuttgart
Projekt digit@L – BOOST. SKILLS. SUPPORT.



Programmierkurs Python

Der Datentyp bool für logische Variablen

Dominik Göddeke

IANS SimTech

Stiftung Innovation in der Hochschullehre

Der Datentyp bool für logische Variablen

Der Datentyp bool für logische Variablen

- Logischer Datentyp: nur zwei Zustände: **wahr** und **falsch**

Der Datentyp bool für logische Variablen

- Logischer Datentyp: nur zwei Zustände: **wahr** und **falsch**
- Auf Hardware-Ebene: **Null oder Eins**, **An oder Aus**

Der Datentyp bool für logische Variablen

- Logischer Datentyp: nur zwei Zustände: **wahr** und **falsch**
- Auf Hardware-Ebene: **Null oder Eins**, **An oder Aus**
- Kann prinzipiell mit einem Bit gespeichert werden

Der Datentyp bool für logische Variablen

- Logischer Datentyp: nur zwei Zustände: **wahr** und **falsch**
- Auf Hardware-Ebene: **Null oder Eins**, **An oder Aus**
- Kann prinzipiell mit einem Bit gespeichert werden
- Offenbar **willkürliche Festlegung**, ob Null=falsch oder andersherum

Der Datentyp bool für logische Variablen

- Logischer Datentyp: nur zwei Zustände: **wahr** und **falsch**
- Auf Hardware-Ebene: **Null oder Eins**, **An oder Aus**
- Kann prinzipiell mit einem Bit gespeichert werden
- Offenbar **willkürliche Festlegung**, ob Null=falsch oder andersherum
- Zur Einordnung

Der Datentyp bool für logische Variablen

- Logischer Datentyp: nur zwei Zustände: **wahr** und **falsch**
- Auf Hardware-Ebene: **Null oder Eins**, **An oder Aus**
- Kann prinzipiell mit einem Bit gespeichert werden
- Offenbar **willkürliche Festlegung**, ob Null=falsch oder andersherum
- Zur Einordnung
  - Ein Bit: kleinste Speichereinheit im Computer

Der Datentyp bool für logische Variablen

- Logischer Datentyp: nur zwei Zustände: **wahr** und **falsch**
- Auf Hardware-Ebene: **Null oder Eins**, **An oder Aus**
- Kann prinzipiell mit einem Bit gespeichert werden
- Offenbar **willkürliche Festlegung**, ob Null=falsch oder andersherum
- Zur Einordnung
  - Ein Bit: kleinste Speichereinheit im Computer
  - 8 Bit formen ein Byte

Der Datentyp bool für logische Variablen

- Logischer Datentyp: nur zwei Zustände: **wahr** und **falsch**
- Auf Hardware-Ebene: **Null oder Eins**, **An oder Aus**
- Kann prinzipiell mit einem Bit gespeichert werden
- Offenbar **willkürliche Festlegung**, ob Null=falsch oder andersherum
- Zur Einordnung
  - Ein Bit: kleinste Speichereinheit im Computer
  - 8 Bit formen ein Byte
  - Monatliches Datenvolumen unserer Mobilfunkverträge, Speicherplatz unserer Telefone im Bereich von (vielen) GigaByte

Der Datentyp bool für logische Variablen

- Logischer Datentyp: nur zwei Zustände: **wahr** und **falsch**
- Auf Hardware-Ebene: **Null oder Eins**, **An oder Aus**
- Kann prinzipiell mit einem Bit gespeichert werden
- Offenbar **willkürliche Festlegung**, ob Null=falsch oder andersherum
- Zur Einordnung
  - Ein Bit: kleinste Speichereinheit im Computer
  - 8 Bit formen ein Byte
  - Monatliches Datenvolumen unserer Mobilfunkverträge, Speicherplatz unserer Telefone im Bereich von (vielen) GigaByte
  - 1 GB sind 1.000.000.000 (eine Milliarde) Byte

Der Datentyp bool für logische Variablen

- Logischer Datentyp: nur zwei Zustände: **wahr** und **falsch**
- Auf Hardware-Ebene: **Null oder Eins**, **An oder Aus**
- Kann prinzipiell mit einem Bit gespeichert werden
- Offenbar **willkürliche Festlegung**, ob Null=falsch oder andersherum
- Zur Einordnung
  - Ein Bit: kleinste Speichereinheit im Computer
  - 8 Bit formen ein Byte
  - Monatliches Datenvolumen unserer Mobilfunkverträge, Speicherplatz unserer Telefone im Bereich von (vielen) GigaByte
  - 1 GB sind 1.000.000.000 (eine Milliarde) Byte
  - Also klar: **ein Bit belegt sehr wenig Speicher**

Der Datentyp bool für logische Variablen

- In Python: spezielle Zustände `True` (wahr / richtig) und `False` (unwahr / falsch)

Der Datentyp bool für logische Variablen

- In Python: spezielle Zustände `True` (wahr / richtig) und `False` (unwahr / falsch)
- Erinnerung: Python unterscheidet zwischen **Groß und Kleinschreibung**

Der Datentyp bool für logische Variablen

- In Python: spezielle Zustände `True` (wahr / richtig) und `False` (unwahr / falsch)
- Erinnerung: Python unterscheidet zwischen **Groß und Kleinschreibung**
  - Deshalb `True`, `False` keine zulässigen Namen für Variablen

Der Datentyp bool für logische Variablen

- In Python: spezielle Zustände `True` (wahr / richtig) und `False` (unwahr / falsch)
- Erinnerung: Python unterscheidet zwischen **Groß und Kleinschreibung**
  - Deshalb `True`, `False` keine zulässigen Namen für Variablen
- Keine Notwendigkeit sich zu merken, ob bspw. `True` gerade Eins oder Null ist

Der Datentyp bool für logische Variablen

- In Python: spezielle Zustände `True` (wahr / richtig) und `False` (unwahr / falsch)
- Erinnerung: Python unterscheidet zwischen **Groß und Kleinschreibung**
  - Deshalb `True`, `False` keine zulässigen Namen für Variablen
- Keine Notwendigkeit sich zu merken, ob bspw. `True` gerade Eins oder Null ist
- Bezeichnung: Datentyp `bool`, **Boolesche Variablen**

Der Datentyp bool für logische Variablen

- In Python: spezielle Zustände `True` (wahr / richtig) und `False` (unwahr / falsch)
- Erinnerung: Python unterscheidet zwischen **Groß und Kleinschreibung**
  - Deshalb `True`, `False` keine zulässigen Namen für Variablen
- Keine Notwendigkeit sich zu merken, ob bspw. `True` gerade Eins oder Null ist
- Bezeichnung: Datentyp `bool`, **Boolesche Variablen**
- George Boole (1815-1864), englischer Mathematiker, Logiker und Philosoph

Boolesche Variablen im Code …

While Schleifen
und Boolesche Variablen

While Schleifen und Boolesche Variablen

```
i = 5
while i >= 1:
  print(i)
  i = i - 1
```

While Schleifen und Boolesche Variablen

```
i = 5
while i >= 1:
  print(i)
  i = i - 1
```

- **Logische Bedingung** der Form `i >= 1`

While Schleifen und Boolesche Variablen

```
i = 5
while i >= 1:
  print(i)
  i = i - 1
```

- **Logische Bedingung** der Form `i >= 1`
- Erinnerung while Schleife: „wiederhole den Block solange die Bedingung erfüllt ist“

While Schleifen und Boolesche Variablen

```
i = 5
while i >= 1:
  print(i)
  i = i - 1
```

- **Logische Bedingung** der Form `i >= 1`
- Erinnerung while Schleife: „wiederhole den Block solange die Bedingung erfüllt ist“
- **Weiter im Code . . .**

Operationen mit Booleschen Variablen

Operationen mit Booleschen Variablen

- Verknüpfung Boolescher Variablen

Operationen mit Booleschen Variablen

- Verknüpfung Boolescher Variablen
  - `not a`: Negation, d.h. `True` wenn `a` `False` ist, `False` wenn `a` `True` ist

Operationen mit Booleschen Variablen

- Verknüpfung Boolescher Variablen
  - `not a`: Negation, d.h. `True` wenn `a` `False` ist, `False` wenn `a` `True` ist
  - `a and b`: `True`, wenn gleichzeitig `a` und `b` `True` sind

Operationen mit Booleschen Variablen

- Verknüpfung Boolescher Variablen
  - `not a`: Negation, d.h. `True` wenn `a` `False` ist, `False` wenn `a` `True` ist
  - `a and b`: `True`, wenn gleichzeitig `a` und `b` `True` sind
  - `a or b`: `True`, wenn `a` und/oder `b` `True` sind

Operationen mit Booleschen Variablen

- Verknüpfung Boolescher Variablen
  - `not a`: Negation, d.h. `True` wenn `a` `False` ist, `False` wenn `a` `True` ist
  - `a and b`: `True`, wenn gleichzeitig `a` und `b` `True` sind
  - `a or b`: `True`, wenn `a` und/oder `b` `True` sind
- **Spielen Sie damit, um ein Gefühl zu bekommen**

Operationen mit Booleschen Variablen

- Verknüpfung Boolescher Variablen
  - `not a`: Negation, d.h. `True` wenn `a` `False` ist, `False` wenn `a` `True` ist
  - `a and b`: `True`, wenn gleichzeitig `a` und `b` `True` sind
  - `a or b`: `True`, wenn `a` und/oder `b` `True` sind
- **Spielen Sie damit, um ein Gefühl zu bekommen**
- Typische Darstellung: **Wahrheitstabellen**, Code im Jupyter

```
  a    |    b   || a and b | a or b
-------+--------++---------+-------
False  | False  ||  False   |  False
False  | True   ||  False   |  True
True   | False  ||  False   |  True
True   | True   ||  True    |  True
```

Operationen mit Booleschen Variablen

- **Reihenfolge der Evaluierungen** (Präzedenzregeln)

Operationen mit Booleschen Variablen

- **Reihenfolge der Evaluierungen** (Präzedenzregeln)
  - `not` vor `and` und `or`

Operationen mit Booleschen Variablen

- **Reihenfolge der Evaluierungen** (Präzedenzregeln)
  - `not` vor `and` und `or`
  - **Klammerung** vor allen

Operationen mit Booleschen Variablen

- **Reihenfolge der Evaluierungen** (Präzedenzregeln)
  - `not` vor `and` und `or`
  - **Klammerung** vor allen
- **Quiz**: Was wird hier ausgegeben?

```
a = True
b = False
print("a =", a)
print("b =", b)
print("not a and b   =", not a and b)
print("not (a and b) =", not (a and b))
print("not a or b    =", not a or b)
print("not (a or b)  =", not (a or b))
```

Operationen mit Booleschen Variablen

- **Reihenfolge der Evaluierungen** (Präzedenzregeln)
  - `not` vor `and` und `or`
  - **Klammerung** vor allen
- **Quiz**: Was wird hier ausgegeben?

```
a = True
b = False
print("a =", a)
print("b =", b)
print("not a and b   =", not a and b)
print("not (a and b) =", not (a and b))
print("not a or b    =", not a or b)
print("not (a or b)  =", not (a or b))
```

- **Hinweis**: Überlegen Sie sich die Antwort anhand der Wahrheitstabellen und Präzedenzregeln

Operationen mit Booleschen Variablen

- **Reihenfolge der Evaluierungen** (Präzedenzregeln)
  - `not` vor `and` und `or`
  - **Klammerung** vor allen
- **Quiz**: Was wird hier ausgegeben?

```
a = True
b = False
print("a =", a)
print("b =", b)
print("not a and b   =", not a and b)
print("not (a and b) =", not (a and b))
print("not a or b    =", not a or b)
print("not (a or b)  =", not (a or b))
```

- **Hinweis**: Überlegen Sie sich die Antwort anhand der Wahrheitstabellen und Präzedenzregeln
- **Auflösung im Code . . .**

De-Morgansche Gesetze

De-Morgansche Gesetze

- Spezifizierung des **„Ausmultiplizierens“ logischer Ausdrücke**

De-Morgansche Gesetze

- Spezifizierung des **„Ausmultiplizierens“ logischer Ausdrücke**
- `https://de.wikipedia.org/wiki/De-morgansche_Gesetze`

De-Morgansche Gesetze

- Spezifizierung des **„Ausmultiplizierens“ logischer Ausdrücke**
- `https://de.wikipedia.org/wiki/De-morgansche_Gesetze`
- Beispiel:

$$\neg(a \wedge b) = \neg a \vee \neg b$$

De-Morgansche Gesetze

- Spezifizierung des **„Ausmultiplizierens“ logischer Ausdrücke**
- `https://de.wikipedia.org/wiki/De-morgansche_Gesetze`
- Beispiel:

$$\neg(a \wedge b) = \neg a \vee \neg b$$

- $\neg$: Negation, `not`

De-Morgansche Gesetze

- Spezifizierung des **„Ausmultiplizierens“ logischer Ausdrücke**
- `https://de.wikipedia.org/wiki/De-morgansche_Gesetze`
- Beispiel:

$$\neg(a \wedge b) = \neg a \vee \neg b$$

- $\neg$: Negation, `not`
- $\wedge$: `and`

De-Morgansche Gesetze

- Spezifizierung des **„Ausmultiplizierens“ logischer Ausdrücke**
- `https://de.wikipedia.org/wiki/De-morgansche_Gesetze`
- Beispiel:

$$\neg(a \wedge b) = \neg a \vee \neg b$$

- $\neg$: Negation, `not`
- $\wedge$: `and`
- $\vee$: `or`

De-Morgansche Gesetze

- Spezifizierung des **„Ausmultiplizierens“ logischer Ausdrücke**
- `https://de.wikipedia.org/wiki/De-morgansche_Gesetze`
- Beispiel:

$$\neg(a \wedge b) = \neg a \vee \neg b$$

- $\neg$: Negation, `not`
- $\wedge$: `and`
- $\vee$: `or`
- **Demo in Jupyter, spielen Sie mit den Werten für `a` und `b`**

Vorausschauende Auswertung

Vorausschauende Auswertung

- Logische Operatoren `and` und `or`: „intelligent“, d.h. „vorausschauend“ in Python

Vorausschauende Auswertung

- Logische Operatoren `and` und `or`: „intelligent“, d.h. „vorausschauend“ in Python
  - `a and b`: wenn `a` bereits zu `False` evaluiert, wird `b` gar nicht erst evaluiert, weil das Ergebnis feststeht

Vorausschauende Auswertung

- Logische Operatoren `and` und `or`: „intelligent“, d.h. „vorausschauend“ in Python
  - `a and b`: wenn `a` bereits zu `False` evaluiert, wird `b` gar nicht erst evaluiert, weil das Ergebnis feststeht
  - `a or b`: wenn `a` bereits wahr ist, wird `b` analog nicht evaluiert

Vorausschauende Auswertung

- Logische Operatoren `and` und `or`: „intelligent“, d.h. „vorausschauend“ in Python
  - `a and b`: wenn `a` bereits zu `False` evaluiert, wird `b` gar nicht erst evaluiert, weil das Ergebnis feststeht
  - `a or b`: wenn `a` bereits wahr ist, wird `b` analog nicht evaluiert
  - **Verifizieren Sie anhand der Wahrheitstabellen, dass dies sinnvoll ist, wenn Ihre Erinnerung an die Mengenlehre und Logik Lücken hat**

Vorausschauende Auswertung

- Logische Operatoren `and` und `or`: „intelligent“, d.h. „vorausschauend“ in Python
  - `a and b`: wenn `a` bereits zu `False` evaluiert, wird `b` gar nicht erst evaluiert, weil das Ergebnis feststeht
  - `a or b`: wenn `a` bereits wahr ist, wird `b` analog nicht evaluiert
  - **Verifizieren Sie anhand der Wahrheitstabellen, dass dies sinnvoll ist, wenn Ihre Erinnerung an die Mengenlehre und Logik Lücken hat**
- Bei komplexeren Ausdrücken extrem sinnvoll

Vorausschauende Auswertung

- Logische Operatoren `and` und `or`: „intelligent“, d.h. „vorausschauend“ in Python
  - `a and b`: wenn `a` bereits zu `False` evaluiert, wird `b` gar nicht erst evaluiert, weil das Ergebnis feststeht
  - `a or b`: wenn `a` bereits wahr ist, wird `b` analog nicht evaluiert
  - **Verifizieren Sie anhand der Wahrheitstabellen, dass dies sinnvoll ist, wenn Ihre Erinnerung an die Mengenlehre und Logik Lücken hat**
- Bei komplexeren Ausdrücken extrem sinnvoll
  - Wenn Auswertung eines logischen Ausdrucks aufwendiger ist als in unseren aktuellen Spielzeugbeispielen

Vorausschauende Auswertung

- Logische Operatoren `and` und `or`: „intelligent“, d.h. „vorausschauend“ in Python
  - `a and b`: wenn `a` bereits zu `False` evaluiert, wird `b` gar nicht erst evaluiert, weil das Ergebnis feststeht
  - `a or b`: wenn `a` bereits wahr ist, wird `b` analog nicht evaluiert
  - **Verifizieren Sie anhand der Wahrheitstabellen, dass dies sinnvoll ist, wenn Ihre Erinnerung an die Mengenlehre und Logik Lücken hat**
- Bei komplexeren Ausdrücken extrem sinnvoll
  - Wenn Auswertung eines logischen Ausdrucks aufwendiger ist als in unseren aktuellen Spielzeugbeispielen
  - Tipp: „einfache“ Bedingungen zuerst überprüfen

Fallstricke bei vorausschauender Auswertung

Fallstricke bei vorausschauender Auswertung

- Fortgeschrittenes Thema

Fallstricke bei vorausschauender Auswertung

- Fortgeschrittenes Thema
- Im zugehörigen Jupyter Notebook bei Interesse

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