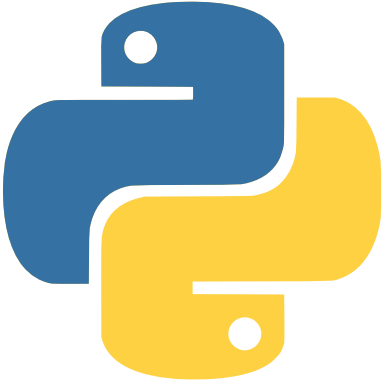
Universität Stuttgart
Projekt digit@L – BOOST. SKILLS. SUPPORT.



Programmierkurs Python

Fehlersuche und Fehlerbehandlung: Unterstützung

Dominik Göddeke

IANS SimTech

Stiftung Innovation in der Hochschullehre

Fehlersuche und Fehlerbehandlung: Unterstützung

Unterstützung bei der Fehlersuche

- Historische Bezeichnung: **bug** für Fehler in einem Programm

Unterstützung bei der Fehlersuche

- Historische Bezeichnung: **bug** für Fehler in einem Programm
- **Debugging**: Prozess der Fehlersuche und -behebung

Unterstützung bei der Fehlersuche

- Historische Bezeichnung: **bug** für Fehler in einem Programm
- **Debugging**: Prozess der Fehlersuche und -behebung
- Dazu: viele Tools, die bei der Fehlersuche helfen

Unterstützung bei der Fehlersuche

- Historische Bezeichnung: **bug** für Fehler in einem Programm
- **Debugging**: Prozess der Fehlersuche und -behebung
- Dazu: viele Tools, die bei der Fehlersuche helfen
- Einfachste Möglichkeit des Debuggings, bekannt seit Lerneinheit 1

Unterstützung bei der Fehlersuche

- Historische Bezeichnung: **bug** für Fehler in einem Programm
- **Debugging**: Prozess der Fehlersuche und -behebung
- Dazu: viele Tools, die bei der Fehlersuche helfen
- Einfachste Möglichkeit des Debuggings, bekannt seit Lerneinheit 1
  - `print()`, oft kombiniert mit `type()` und `id()`

Unterstützung bei der Fehlersuche

- Historische Bezeichnung: **bug** für Fehler in einem Programm
- **Debugging**: Prozess der Fehlersuche und -behebung
- Dazu: viele Tools, die bei der Fehlersuche helfen
- Einfachste Möglichkeit des Debuggings, bekannt seit Lerneinheit 1
  - `print()`, oft kombiniert mit `type()` und `id()`
  - Unprofessionell: viele zusätzliche Codezeilen, ein- und auskommentiert

Unterstützung bei der Fehlersuche

- Historische Bezeichnung: **bug** für Fehler in einem Programm
- **Debugging**: Prozess der Fehlersuche und -behebung
- Dazu: viele Tools, die bei der Fehlersuche helfen
- Einfachste Möglichkeit des Debuggings, bekannt seit Lerneinheit 1
  - `print()`, oft kombiniert mit `type()` und `id()`
  - Unprofessionell: viele zusätzliche Codezeilen, ein- und auskommentiert
  - Unschlau: Löschen der Debug-Ausgaben impliziert Neuprogrammierung, wenn Fehler erneut auftritt

Unterstützung bei der Fehlersuche

- Historische Bezeichnung: **bug** für Fehler in einem Programm
- **Debugging**: Prozess der Fehlersuche und -behebung
- Dazu: viele Tools, die bei der Fehlersuche helfen
- Einfachste Möglichkeit des Debuggings, bekannt seit Lerneinheit 1
  - `print()`, oft kombiniert mit `type()` und `id()`
  - Unprofessionell: viele zusätzliche Codezeilen, ein- und auskommentiert
  - Unschlau: Löschen der Debug-Ausgaben impliziert Neuprogrammierung, wenn Fehler erneut auftritt
- **Ziel** nun: schlauere Möglichkeit, insb. bei inkrementeller Programmentwicklung

Assertions

Assertions

- Deutlich besser: `assert`

Assertions

- Deutlich besser: `assert`
- Syntax

```
assert some_condition
assert some_condition, 'explanation'
```

Assertions

- Deutlich besser: `assert`
- Syntax

```
assert some_condition
assert some_condition, 'explanation'
```

- `some_condition == True`: es passiert gar nichts

Assertions

- Deutlich besser: `assert`
- Syntax

```
assert some_condition
assert some_condition, 'explanation'
```

- `some_condition == True`: es passiert gar nichts
- Andernfalls: Exception vom Typ `AssertionError`

Codebeispiele

Anwendungsbeispiele für assertions

Anwendungsbeispiele für assertions

- Beispiel schlecht gewählt: Validierung von Benutzerangaben nicht mit `assert`

Anwendungsbeispiele für assertions

- Beispiel schlecht gewählt: Validierung von Benutzerangaben nicht mit `assert`
- **Stärke von assertions**: Dokumentation unserer Denkweise und Logik bei der Erstellung eines Programms

Anwendungsbeispiele für assertions

- Beispiel schlecht gewählt: Validierung von Benutzerangaben nicht mit `assert`
- **Stärke von assertions**: Dokumentation unserer Denkweise und Logik bei der Erstellung eines Programms
  - Indexarithmetik-Überlegungen: Ergebnis Indexrechnung immer gerade Zahl

Anwendungsbeispiele für assertions

- Beispiel schlecht gewählt: Validierung von Benutzerangaben nicht mit `assert`
- **Stärke von assertions**: Dokumentation unserer Denkweise und Logik bei der Erstellung eines Programms
  - Indexarithmetik-Überlegungen: Ergebnis Indexrechnung immer gerade Zahl
  - Physik-Überlegungen: Größe immer positiv

Anwendungsbeispiele für assertions

- Beispiel schlecht gewählt: Validierung von Benutzerangaben nicht mit `assert`
- **Stärke von assertions**: Dokumentation unserer Denkweise und Logik bei der Erstellung eines Programms
  - Indexarithmetik-Überlegungen: Ergebnis Indexrechnung immer gerade Zahl
  - Physik-Überlegungen: Größe immer positiv
- Assertion erlaubt Prüfung solcher **Invarianten** ohne negativen Einfluss auf Code, und Vermeidung unleserlicher `IndexError`

Anwendungsbeispiele für assertions

- Beispiel schlecht gewählt: Validierung von Benutzerangaben nicht mit `assert`
- **Stärke von assertions**: Dokumentation unserer Denkweise und Logik bei der Erstellung eines Programms
  - Indexarithmetik-Überlegungen: Ergebnis Indexrechnung immer gerade Zahl
  - Physik-Überlegungen: Größe immer positiv
- Assertion erlaubt Prüfung solcher **Invarianten** ohne negativen Einfluss auf Code, und Vermeidung unleserlicher `IndexError`
- **Daumenregel**

Anwendungsbeispiele für assertions

- Beispiel schlecht gewählt: Validierung von Benutzerangaben nicht mit `assert`
- **Stärke von assertions**: Dokumentation unserer Denkweise und Logik bei der Erstellung eines Programms
  - Indexarithmetik-Überlegungen: Ergebnis Indexrechnung immer gerade Zahl
  - Physik-Überlegungen: Größe immer positiv
- Assertion erlaubt Prüfung solcher **Invarianten** ohne negativen Einfluss auf Code, und Vermeidung unleserlicher `IndexError`
- **Daumenregel**
  - `try/except` Blöcke: hilfreicher bei Fehlern der Benutzer*innen

Anwendungsbeispiele für assertions

- Beispiel schlecht gewählt: Validierung von Benutzerangaben nicht mit `assert`
- **Stärke von assertions**: Dokumentation unserer Denkweise und Logik bei der Erstellung eines Programms
  - Indexarithmetik-Überlegungen: Ergebnis Indexrechnung immer gerade Zahl
  - Physik-Überlegungen: Größe immer positiv
- Assertion erlaubt Prüfung solcher **Invarianten** ohne negativen Einfluss auf Code, und Vermeidung unleserlicher `IndexError`
- **Daumenregel**
  - `try/except` Blöcke: hilfreicher bei Fehlern der Benutzer*innen
  - `assert`: hilfreicher zur Vermeidung der Wiederholung eigener Fehler

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