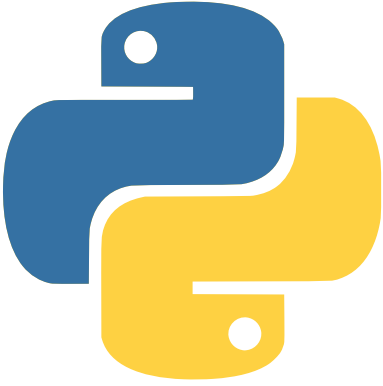
Universität Stuttgart
Projekt digit@L – BOOST. SKILLS. SUPPORT.



Programmierkurs Python

Konstruktoren und magic methods

Dominik Göddeke

IANS SimTech

Stiftung Innovation in der Hochschullehre

Konstruktoren und magic methods

Konstruktoren

Konstruktoren

- `__init__` Methode: Initialisierung eines Objekts

Konstruktoren

- `__init__` Methode: Initialisierung eines Objekts
- Sprechweise in OOP: **Konstruktor**

Konstruktoren

- `__init__` Methode: Initialisierung eines Objekts
- Sprechweise in OOP: **Konstruktor**
- Funktion konstruiert ein Objekt der Klasse

Konstruktoren

- `__init__` Methode: Initialisierung eines Objekts
- Sprechweise in OOP: **Konstruktor**
- Funktion konstruiert ein Objekt der Klasse
- Signatur: `__init__(self[, ...])`

Konstruktoren

- `__init__` Methode: Initialisierung eines Objekts
- Sprechweise in OOP: **Konstruktor**
- Funktion konstruiert ein Objekt der Klasse
- Signatur: `__init__(self[, ...])`
- Verwendungszweck: Definition von Attributen, Initialisierung mit Standardwerten

Konstruktoren

- `__init__` Methode: Initialisierung eines Objekts
- Sprechweise in OOP: **Konstruktor**
- Funktion konstruiert ein Objekt der Klasse
- Signatur: `__init__(self[, ...])`
- Verwendungszweck: Definition von Attributen, Initialisierung mit Standardwerten
- „Beste“ Möglichkeit: optionale Default-Argumente, verschiedene Konstruktoren möglich

Konstruktoren

- `__init__` Methode: Initialisierung eines Objekts
- Sprechweise in OOP: **Konstruktor**
- Funktion konstruiert ein Objekt der Klasse
- Signatur: `__init__(self[, ...])`
- Verwendungszweck: Definition von Attributen, Initialisierung mit Standardwerten
- „Beste“ Möglichkeit: optionale Default-Argumente, verschiedene Konstruktoren möglich
- Technisch immer noch möglich, andere Attribute anderswo zu definieren

Konstruktoren

- `__init__` Methode: Initialisierung eines Objekts
- Sprechweise in OOP: **Konstruktor**
- Funktion konstruiert ein Objekt der Klasse
- Signatur: `__init__(self[, ...])`
- Verwendungszweck: Definition von Attributen, Initialisierung mit Standardwerten
- „Beste“ Möglichkeit: optionale Default-Argumente, verschiedene Konstruktoren möglich
- Technisch immer noch möglich, andere Attribute anderswo zu definieren
- Konterkariert Kapselung

Konstruktoren

- `__init__` Methode: Initialisierung eines Objekts
- Sprechweise in OOP: **Konstruktor**
- Funktion konstruiert ein Objekt der Klasse
- Signatur: `__init__(self[, ...])`
- Verwendungszweck: Definition von Attributen, Initialisierung mit Standardwerten
- „Beste“ Möglichkeit: optionale Default-Argumente, verschiedene Konstruktoren möglich
- Technisch immer noch möglich, andere Attribute anderswo zu definieren
- Konterkariert Kapselung
- Später: Möglichkeit zur Verhinderung

Codebeispiele

Magische Methoden und Operator Overloading

Magische Methoden und Operator Overloading

- Magic methods: Methoden einer Klasse, die nie explizit aufgerufen werden

Magische Methoden und Operator Overloading

- Magic methods: Methoden einer Klasse, die nie explizit aufgerufen werden
- Am besten erklärt durch Beispiele

Magische Methoden und Operator Overloading

- Magic methods: Methoden einer Klasse, die nie explizit aufgerufen werden
- Am besten erklärt durch Beispiele
  - `__str__(self)`: String-Repräsentation

Magische Methoden und Operator Overloading

- Magic methods: Methoden einer Klasse, die nie explizit aufgerufen werden
- Am besten erklärt durch Beispiele
  - `__str__(self)`: String-Repräsentation
  - `__add__(self, other)`: Addition

Magische Methoden und Operator Overloading

- Magic methods: Methoden einer Klasse, die nie explizit aufgerufen werden
- Am besten erklärt durch Beispiele
  - `__str__(self)`: String-Repräsentation
  - `__add__(self, other)`: Addition
  - `__lt__(self, other)`: Vergleich $\leq$

Magische Methoden und Operator Overloading

- Magic methods: Methoden einer Klasse, die nie explizit aufgerufen werden
- Am besten erklärt durch Beispiele
  - `__str__(self)`: String-Repräsentation
  - `__add__(self, other)`: Addition
  - `__lt__(self, other)`: Vergleich $\leq$
  - `__call__(self, args)` Auswertung

Magische Methoden und Operator Overloading

- Magic methods: Methoden einer Klasse, die nie explizit aufgerufen werden
- Am besten erklärt durch Beispiele
  - `__str__(self)`: String-Repräsentation
  - `__add__(self, other)`: Addition
  - `__lt__(self, other)`: Vergleich $\leq$
  - `__call__(self, args)` Auswertung
- Jetzt klar: `__str()__` automatisch aufgerufen bei `print()`

Magische Methoden und Operator Overloading

- Magic methods: Methoden einer Klasse, die nie explizit aufgerufen werden
- Am besten erklärt durch Beispiele
  - `__str__(self)`: String-Repräsentation
  - `__add__(self, other)`: Addition
  - `__lt__(self, other)`: Vergleich $\leq$
  - `__call__(self, args)` Auswertung
- Jetzt klar: `__str()__` automatisch aufgerufen bei `print()`
- `__add()__` erklärt `a+b` für Zahltypen und Zeichenketten

Magische Methoden und Operator Overloading

- Magic methods: Methoden einer Klasse, die nie explizit aufgerufen werden
- Am besten erklärt durch Beispiele
  - `__str__(self)`: String-Repräsentation
  - `__add__(self, other)`: Addition
  - `__lt__(self, other)`: Vergleich $\leq$
  - `__call__(self, args)` Auswertung
- Jetzt klar: `__str()__` automatisch aufgerufen bei `print()`
- `__add()__` erklärt `a+b` für Zahltypen und Zeichenketten
- Sprechweise im OOP: **operator overloading**, etablierter Operator wie `+-*/` erhält neue Bedeutung für Instanzen einer Klasse

Codebeispiele

Sichtbarkeit von Variablen

Sichtbarkeit von Variablen

- Bisher: Attribute global sichtbar und veränderbar

Sichtbarkeit von Variablen

- Bisher: Attribute global sichtbar und veränderbar
- Daten also noch **nicht vollständig gekapselt**

Sichtbarkeit von Variablen

- Bisher: Attribute global sichtbar und veränderbar
- Daten also noch **nicht vollständig gekapselt**
- Lösung: **private Attribute**

Sichtbarkeit von Variablen

- Bisher: Attribute global sichtbar und veränderbar
- Daten also noch **nicht vollständig gekapselt**
- Lösung: **private Attribute**
- In Python: **doppelte Unterstriche** zu Beginn des Variablennamens

Sichtbarkeit von Variablen

- Bisher: Attribute global sichtbar und veränderbar
- Daten also noch **nicht vollständig gekapselt**
- Lösung: **private Attribute**
- In Python: **doppelte Unterstriche** zu Beginn des Variablennamens
- Später bei Vererbung: oft **einfache Unterstriche** schlauer

Sichtbarkeit von Variablen

- Bisher: Attribute global sichtbar und veränderbar
- Daten also noch **nicht vollständig gekapselt**
- Lösung: **private Attribute**
- In Python: **doppelte Unterstriche** zu Beginn des Variablennamens
- Später bei Vererbung: oft **einfache Unterstriche** schlauer
- Lese- und Schreiboperationen für private Attribute: explizit als Methode

Codebeispiele

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