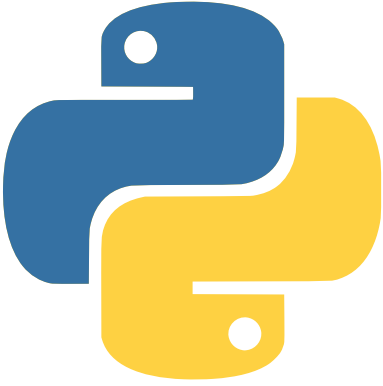
Universität Stuttgart
Projekt digit@L – BOOST. SKILLS. SUPPORT.



Programmierkurs Python

Funktionen: Einführung

Dominik Göddeke

IANS SimTech

Stiftung Innovation in der Hochschullehre

Funktionen: Einführung

Funktionen

- Ziel dieser 3 Videos: **Funktionen** in Python

Funktionen

- Ziel dieser 3 Videos: **Funktionen** in Python
- Klar: Mathematische Funktionen $f : A \to B$

Funktionen

- Ziel dieser 3 Videos: **Funktionen** in Python
- Klar: Mathematische Funktionen $f : A \to B$
- Aber viel mehr, insb. **Vermeidung von Code-Duplizierung** durch Parametrisierung

Start mit einem Negativbeispiel

Eine erste Funktion

Eine erste Funktion

- Idee von **Python-Funktionen**

Eine erste Funktion

- Idee von **Python-Funktionen**
  - Wiederverwendbare Code-Bestandteile nur einmal programmieren

Eine erste Funktion

- Idee von **Python-Funktionen**
  - Wiederverwendbare Code-Bestandteile nur einmal programmieren
  - Verwendung von Code-Bestandteilen in verschiedenen Kontexten (an verschiedenen Stellen des Programms)

Eine erste Funktion

- Idee von **Python-Funktionen**
  - Wiederverwendbare Code-Bestandteile nur einmal programmieren
  - Verwendung von Code-Bestandteilen in verschiedenen Kontexten (an verschiedenen Stellen des Programms)
- Nun: inkrementelle Diskussion des **Syntax** von Funktionen

Eine erste Funktion

- Idee von **Python-Funktionen**
  - Wiederverwendbare Code-Bestandteile nur einmal programmieren
  - Verwendung von Code-Bestandteilen in verschiedenen Kontexten (an verschiedenen Stellen des Programms)
- Nun: inkrementelle Diskussion des **Syntax** von Funktionen
- Und Rolle im **Programmfluss**

Eine erste Funktion

- Idee von **Python-Funktionen**
  - Wiederverwendbare Code-Bestandteile nur einmal programmieren
  - Verwendung von Code-Bestandteilen in verschiedenen Kontexten (an verschiedenen Stellen des Programms)
- Nun: inkrementelle Diskussion des **Syntax** von Funktionen
- Und Rolle im **Programmfluss**
- `def` Anweisung definiert Funktionen

Eine erste Funktion

- Idee von **Python-Funktionen**
  - Wiederverwendbare Code-Bestandteile nur einmal programmieren
  - Verwendung von Code-Bestandteilen in verschiedenen Kontexten (an verschiedenen Stellen des Programms)
- Nun: inkrementelle Diskussion des **Syntax** von Funktionen
- Und Rolle im **Programmfluss**
- `def` Anweisung definiert Funktionen
- Gefolgt von Block, d.h. Doppelpunkt und Einrückung

Wie funktioniert das im Code?

Wichtig für das Verständnis

Wichtig für das Verständnis

- Bisher schon viele Funktionsaufrufe

Wichtig für das Verständnis

- Bisher schon viele Funktionsaufrufe
- Ohne es explizit zu benennen

Wichtig für das Verständnis

- Bisher schon viele Funktionsaufrufe
- Ohne es explizit zu benennen
  - `print()`, `id()`, `type()`, `range()`, `math.cos()`, ...

Wichtig für das Verständnis

- Bisher schon viele Funktionsaufrufe
- Ohne es explizit zu benennen
  - `print()`, `id()`, `type()`, `range()`, `math.cos()`, ...
- **Überladung** von Funktionen

Wichtig für das Verständnis

- Bisher schon viele Funktionsaufrufe
- Ohne es explizit zu benennen
  - `print()`, `id()`, `type()`, `range()`, `math.cos()`, ...
- **Überladung** von Funktionen
  - Fachbegriff für „überschreiben“

Wichtig für das Verständnis

- Bisher schon viele Funktionsaufrufe
- Ohne es explizit zu benennen
  - `print()`, `id()`, `type()`, `range()`, `math.cos()`, ...
- **Überladung** von Funktionen
  - Fachbegriff für „überschreiben“
  - Relevant: letzte Definition

Wichtig für das Verständnis

- Bisher schon viele Funktionsaufrufe
- Ohne es explizit zu benennen
  - `print()`, `id()`, `type()`, `range()`, `math.cos()`, ...
- **Überladung** von Funktionen
  - Fachbegriff für „überschreiben“
  - Relevant: letzte Definition
  - Sollte man vermeiden

Codebeispiel zur Überladung

Programmfluss
Wir spielen Interpreter

Programmfluss – Wir spielen Interpreter

```
import math

def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

Programmfluss – Wir spielen Interpreter

```
import math

def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Verständnis von Funktionen erfordert Verständnis des Programmfluss

Programmfluss – Wir spielen Interpreter

```
import math

def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Verständnis von Funktionen erfordert Verständnis des Programmfluss
- Wir „spielen“ die Rolle des Python-Interpreters für dieses Beispiel

Programmfluss – Wir spielen Interpreter

```
import math

def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Verständnis von Funktionen erfordert Verständnis des Programmfluss
- Wir „spielen“ die Rolle des Python-Interpreters für dieses Beispiel
- Wichtig: Interpreter marschiert zeilenweise (Anweisungs-weise) durch den Code

Programmfluss – Wir spielen Interpreter

```
import math

def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

Programmfluss – Wir spielen Interpreter

```
import math

def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Zeile `import math`: Bibliothek `math` wird geladen

Programmfluss – Wir spielen Interpreter

```
import math

def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Zeile `import math`: Bibliothek `math` wird geladen
- Später: Funktionsweise von Bibliotheken, hier „Magie“

Programmfluss – Wir spielen Interpreter

```
import math

def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Zeile `import math`: Bibliothek `math` wird geladen
- Später: Funktionsweise von Bibliotheken, hier „Magie“
- Überspringen der Leerzeile

Programmfluss – Wir spielen Interpreter

```
def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

Programmfluss – Wir spielen Interpreter

```
def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Zeile `def ...`: Definition der Funktion `print_square_root_of_pi()`

Programmfluss – Wir spielen Interpreter

```
def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Zeile `def ...`: Definition der Funktion `print_square_root_of_pi()`
- Code innerhalb der Funktionsdefinition wird **nicht** ausgeführt

Programmfluss – Wir spielen Interpreter

```
def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Zeile `def ...`: Definition der Funktion `print_square_root_of_pi()`
- Code innerhalb der Funktionsdefinition wird **nicht** ausgeführt
- Stattdessen: Interpreter „merkt“ sich Existenz der Funktion

Programmfluss – Wir spielen Interpreter

```
def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Zeile `def ...`: Definition der Funktion `print_square_root_of_pi()`
- Code innerhalb der Funktionsdefinition wird **nicht** ausgeführt
- Stattdessen: Interpreter „merkt“ sich Existenz der Funktion
  - Name: `print_square_root_of_pi()`

Programmfluss – Wir spielen Interpreter

```
def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Zeile `def ...`: Definition der Funktion `print_square_root_of_pi()`
- Code innerhalb der Funktionsdefinition wird **nicht** ausgeführt
- Stattdessen: Interpreter „merkt“ sich Existenz der Funktion
  - Name: `print_square_root_of_pi()`
  - Stelle im Speicher, wo Funktionscode (entsprechender Block) liegt

Programmfluss – Wir spielen Interpreter

```
def print_square_root_of_pi():
    print(math.sqrt(math.pi))

# Aufruf der Funktion:
print_square_root_of_pi()
```

- Zeile `def ...`: Definition der Funktion `print_square_root_of_pi()`
- Code innerhalb der Funktionsdefinition wird **nicht** ausgeführt
- Stattdessen: Interpreter „merkt“ sich Existenz der Funktion
  - Name: `print_square_root_of_pi()`
  - Stelle im Speicher, wo Funktionscode (entsprechender Block) liegt
- Wieder: Überspringen der Leerzeile

Programmfluss – Wir spielen Interpreter

```
# Aufruf der Funktion:
print_square_root_of_pi()
```

Programmfluss – Wir spielen Interpreter

```
# Aufruf der Funktion:
print_square_root_of_pi()
```

- Ignorieren des Kommentars

Programmfluss – Wir spielen Interpreter

```
# Aufruf der Funktion:
print_square_root_of_pi()
```

- Ignorieren des Kommentars
- Dann endlich: **Ausführung der Funktion** `print_square_root_of_pi()`

Programmfluss – Wir spielen Interpreter

```
# Aufruf der Funktion:
print_square_root_of_pi()
```

- Ignorieren des Kommentars
- Dann endlich: **Ausführung der Funktion** `print_square_root_of_pi()`
- Interpreter springt in den Block der Funktion

Programmfluss – Wir spielen Interpreter

```
# Aufruf der Funktion:
print_square_root_of_pi()
```

- Ignorieren des Kommentars
- Dann endlich: **Ausführung der Funktion** `print_square_root_of_pi()`
- Interpreter springt in den Block der Funktion
- Interpreter führt die dortigen Anweisungen aus

Programmfluss – Wir spielen Interpreter

```
# Aufruf der Funktion:
print_square_root_of_pi()
```

- Ignorieren des Kommentars
- Dann endlich: **Ausführung der Funktion** `print_square_root_of_pi()`
- Interpreter springt in den Block der Funktion
- Interpreter führt die dortigen Anweisungen aus
- Danach weiter nach dem Funktionsaufruf, im Beispiel Programmende

Quizzes

Quizzes

```
import math

print_square_root_of_e()

def print_square_root_of_e():
    print(math.sqrt(math.exp(1)))
```

Quizzes

```
import math

print_square_root_of_e()

def print_square_root_of_e():
    print(math.sqrt(math.exp(1)))
```

- Quiz: Warum funktioniert dieser Code nicht?

Quizzes

```
import math

print_square_root_of_e()

def print_square_root_of_e():
    print(math.sqrt(math.exp(1)))
```

- Quiz: Warum funktioniert dieser Code nicht?
- Lösung: Interpreter kennt **Funktionsaufruf** vor **Funktionsdefinition** nicht

Quizzes

```
def hello_world():
    print("Hello world!")
    print_my_name()

def print_my_name():
    name = input("Dein Name: ")
    print("Und ein herzliches Willkommen an", name)

hello_world()
```

Quizzes

```
def hello_world():
    print("Hello world!")
    print_my_name()

def print_my_name():
    name = input("Dein Name: ")
    print("Und ein herzliches Willkommen an", name)

hello_world()
```

- Quiz: Warum funktioniert dieser Code?

Quizzes

```
def hello_world():
    print("Hello world!")
    print_my_name()

def print_my_name():
    name = input("Dein Name: ")
    print("Und ein herzliches Willkommen an", name)

hello_world()
```

- Quiz: Warum funktioniert dieser Code?
- Lösung: Interpreter kennt **Funktionsaufruf** vor **Funktionsdefinition**

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