

Informatik I: Einführung in die Programmierung

10. Programmentwicklung: Testen und Debuggen

Albert-Ludwigs-Universität Freiburg



UNI
FREIBURG

Peter Thiemann

11. Dezember 2019



Programmmentwicklung

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Wie kommen Fehler ins Programm?



- Beim Schreiben von Programmen gelingt nicht alles auf Anhieb.

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Wie kommen Fehler ins Programm?



- Beim Schreiben von Programmen gelingt nicht alles auf Anhieb.
- Besonders in einfach erscheinenden Fällen: Schreibfehler, zu kurz gedacht, falsche Annahmen, ...

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Wie kommen Fehler ins Programm?



- Beim Schreiben von Programmen gelingt nicht alles auf Anhieb.
- Besonders in einfach erscheinenden Fällen: Schreibfehler, zu kurz gedacht, falsche Annahmen, ...
- “Rund **50%** des Programmieraufwands wird für die Identifikation und Beseitigung von Fehlern aufgewendet.”

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Wie kommen Fehler ins Programm?



- Beim Schreiben von Programmen gelingt nicht alles auf Anhieb.
- Besonders in einfach erscheinenden Fällen: Schreibfehler, zu kurz gedacht, falsche Annahmen, ...
- “Rund **50%** des Programmieraufwands wird für die Identifikation und Beseitigung von Fehlern aufgewendet.”
- “The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.” — Tom Cargill, Bell Labs

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Wie kommen Fehler ins Programm?



- Beim Schreiben von Programmen gelingt nicht alles auf Anhieb.
- Besonders in einfach erscheinenden Fällen: Schreibfehler, zu kurz gedacht, falsche Annahmen, ...
- “Rund **50%** des Programmieraufwands wird für die Identifikation und Beseitigung von Fehlern aufgewendet.”
- “The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.” — Tom Cargill, Bell Labs
- Auch “fertige” Software hat noch **1–18 Fehler pro 1000 Zeilen Code!**

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Wie kommen Fehler ins Programm?



- Beim Schreiben von Programmen gelingt nicht alles auf Anhieb.
- Besonders in einfach erscheinenden Fällen: Schreibfehler, zu kurz gedacht, falsche Annahmen, ...
- “Rund **50%** des Programmieraufwands wird für die Identifikation und Beseitigung von Fehlern aufgewendet.”
- “The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.” — Tom Cargill, Bell Labs
- Auch “fertige” Software hat noch **1–18 Fehler pro 1000 Zeilen Code!**
- Wichtig: **Werkzeuge** für die Fehlersuche und für die Qualitätskontrolle

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Wie kommen Fehler ins Programm?



- Beim Schreiben von Programmen gelingt nicht alles auf Anhieb.
- Besonders in einfach erscheinenden Fällen: Schreibfehler, zu kurz gedacht, falsche Annahmen, ...
- “Rund **50%** des Programmieraufwands wird für die Identifikation und Beseitigung von Fehlern aufgewendet.”
- “The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.” — Tom Cargill, Bell Labs
- Auch “fertige” Software hat noch **1–18 Fehler pro 1000 Zeilen Code!**
- Wichtig: **Werkzeuge** für die Fehlersuche und für die Qualitätskontrolle
- Erster Schritt dazu: **automatisches Testen**

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Wir wollen ein Programm entwickeln, das den Wert eines arithmetischen Ausdrucks, repräsentiert durch einen Ausdrucksbaum, über den ganzen Zahlen errechnet.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Wir wollen ein Programm entwickeln, das den Wert eines arithmetischen Ausdrucks, repräsentiert durch einen Ausdrucksbaum, über den ganzen Zahlen errechnet.
- Beispiel: $\text{Node} ('*', \text{Node} ('+', \text{leaf}(2), \text{leaf}(5)), \text{leaf}(6)) \mapsto 42$

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Wir wollen ein Programm entwickeln, das den Wert eines arithmetischen Ausdrucks, repräsentiert durch einen Ausdrucksbaum, über den ganzen Zahlen errechnet.
- Beispiel: $\text{Node} ('*', \text{Node} ('+', \text{leaf}(2), \text{leaf}(5)), \text{leaf}(6)) \mapsto 42$
- Methode: Induktive Traversierung des Ausdrucksbaums.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Wir wollen ein Programm entwickeln, das den Wert eines arithmetischen Ausdrucks, repräsentiert durch einen Ausdrucksbaum, über den ganzen Zahlen errechnet.
- Beispiel: $\text{Node} ('*', \text{Node} ('+', \text{leaf}(2), \text{leaf}(5)), \text{leaf}(6)) \mapsto 42$
- Methode: Induktive Traversierung des Ausdrucksbaums.
- Annahme: der Baum ist nicht leer

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



```
def exprval(tree : Node) -> int
  if tree.mark == '+':
    return exprval(tree.left)+exprval(tree.right)
  elif tree.mark == '-':
    return exprval(tree.left)-exprval(tree.right)
  elif tree.mark == '*':
    return exprval(tree.left)*exprval(tree.right)
  elif tree.mark == '/':
    return exprval(tree.left)/exprval(tree.right))
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Syntaxfehler

Das Programm entspricht nicht der formalen Grammatik. Solche Fehler bemerkt der Python-Interpreter vor der Ausführung. Sie sind meist einfach zu finden und zu reparieren.

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Syntaxfehler

Das Programm entspricht nicht der formalen Grammatik. Solche Fehler bemerkt der Python-Interpreter vor der Ausführung. Sie sind meist einfach zu finden und zu reparieren.

Laufzeitfehler

Während der Ausführung passiert nichts (das Programm hängt) oder es gibt eine Fehlermeldung (**Exception**).

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Syntaxfehler

Das Programm entspricht nicht der formalen Grammatik. Solche Fehler bemerkt der Python-Interpreter vor der Ausführung. Sie sind meist einfach zu finden und zu reparieren.

Laufzeitfehler

Während der Ausführung passiert nichts (das Programm hängt) oder es gibt eine Fehlermeldung (**Exception**).

Logische Fehler

Alles „läuft“, aber die Ausgaben und Aktionen des Programms sind anders als erwartet. Das sind die gefährlichsten Fehler. Beispiel: *Mars-Climate-Orbiter*.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler
Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)
- Das tatsächliche Problem kann aber mehrere Zeilen vorher liegen!

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)
- Das tatsächliche Problem kann aber mehrere Zeilen vorher liegen!
- Typische mögliche Fehler:

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)
- Das tatsächliche Problem kann aber mehrere Zeilen vorher liegen!
- Typische mögliche Fehler:
 - Schlüsselwort als Variablennamen benutzt

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)
- Das tatsächliche Problem kann aber mehrere Zeilen vorher liegen!
- Typische mögliche Fehler:
 - Schlüsselwort als Variablennamen benutzt
 - Es fehlt ein ':' für ein mehrzeiliges Statement (`while`, `if`, `for`, `def`, usw.)

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)
- Das tatsächliche Problem kann aber mehrere Zeilen vorher liegen!
- Typische mögliche Fehler:
 - Schlüsselwort als Variablennamen benutzt
 - Es fehlt ein ':' für ein mehrzeiliges Statement (`while`, `if`, `for`, `def`, usw.)
 - Nicht abgeschlossener Multi-Zeilen-String (drei öffnende Anführungszeichen)

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)
- Das tatsächliche Problem kann aber mehrere Zeilen vorher liegen!
- Typische mögliche Fehler:
 - Schlüsselwort als Variablennamen benutzt
 - Es fehlt ein ':' für ein mehrzeiliges Statement (`while`, `if`, `for`, `def`, usw.)
 - Nicht abgeschlossener Multi-Zeilen-String (drei öffnende Anführungszeichen)
 - Unbalancierte Klammern

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)
- Das tatsächliche Problem kann aber mehrere Zeilen vorher liegen!
- Typische mögliche Fehler:
 - Schlüsselwort als Variablennamen benutzt
 - Es fehlt ein ':' für ein mehrzeiliges Statement (`while`, `if`, `for`, `def`, usw.)
 - Nicht abgeschlossener Multi-Zeilen-String (drei öffnende Anführungszeichen)
 - Unbalancierte Klammern
 - `=` statt `==` in Booleschen Ausdrücken

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)
- Das tatsächliche Problem kann aber mehrere Zeilen vorher liegen!
- Typische mögliche Fehler:
 - Schlüsselwort als Variablennamen benutzt
 - Es fehlt ein ':' für ein mehrzeiliges Statement (`while`, `if`, `for`, `def`, usw.)
 - Nicht abgeschlossener Multi-Zeilen-String (drei öffnende Anführungszeichen)
 - Unbalancierte Klammern
 - `=` statt `==` in Booleschen Ausdrücken
 - Die Einrückung!

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)
- Das tatsächliche Problem kann aber mehrere Zeilen vorher liegen!
- Typische mögliche Fehler:
 - Schlüsselwort als Variablennamen benutzt
 - Es fehlt ein ':' für ein mehrzeiliges Statement (`while`, `if`, `for`, `def`, usw.)
 - Nicht abgeschlossener Multi-Zeilen-String (drei öffnende Anführungszeichen)
 - Unbalancierte Klammern
 - `=` statt `==` in Booleschen Ausdrücken
 - Die Einrückung!
- Oft helfen Editoren mit Syntaxunterstützung.

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler festgestellt wurde (in IDLE wird die Zeile markiert)
- Das tatsächliche Problem kann aber mehrere Zeilen vorher liegen!
- Typische mögliche Fehler:
 - Schlüsselwort als Variablennamen benutzt
 - Es fehlt ein ':' für ein mehrzeiliges Statement (`while`, `if`, `for`, `def`, usw.)
 - Nicht abgeschlossener Multi-Zeilen-String (drei öffnende Anführungszeichen)
 - Unbalancierte Klammern
 - `=` statt `==` in Booleschen Ausdrücken
 - Die Einrückung!
- Oft helfen Editoren mit Syntaxunterstützung.
- Im schlechtesten Fall: Sukzessives Auskommentieren und Probieren

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Unser Programm enthält 2 Syntaxfehler.

Evaluating an Expression tree

```
def exprval(tree : Node) -> int
  if tree.mark == '+':
    return exprval(tree.left)+exprval(tree.right)
  elif tree.mark == '-':
    return exprval(tree.left)-exprval(tree.right)
  elif tree.mark == '*':
    return exprval(tree.left)*exprval(tree.right)
  elif tree.mark == '/':
    return exprval(tree.left)/exprval(tree.right))
```

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Unser Programm enthält 2 Syntaxfehler.
- Das syntaktisch korrekte Programm:

Evaluating an Expression tree

```
def exprval(tree : Node) -> int:  
  if tree.mark == '+':  
    return exprval(tree.left)+exprval(tree.right)  
  elif tree.mark == '-':  
    return exprval(tree.left)-exprval(tree.right)  
  elif tree.mark == '*':  
    return exprval(tree.left)*exprval(tree.right)  
  elif tree.mark == '/':  
    return exprval(tree.left)/exprval(tree.right) )
```

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Das Programm wartet auf eine Eingabe (→ kein Fehler, Eingabe machen).

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Laufzeitfehler: Das Programm „hängt“



- Das Programm wartet auf eine Eingabe (→ kein Fehler, Eingabe machen).
- Es wartet auf Daten aus anderer Quelle (ggfs. Timeout vorsehen).

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Laufzeitfehler: Das Programm „hängt“



- Das Programm wartet auf eine Eingabe (→ kein Fehler, Eingabe machen).
- Es wartet auf Daten aus anderer Quelle (ggfs. Timeout vorsehen).
- Es befindet sich in einer **Endlosschleife**.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Das Programm wartet auf eine Eingabe (→ kein Fehler, Eingabe machen).
- Es wartet auf Daten aus anderer Quelle (ggfs. Timeout vorsehen).
- Es befindet sich in einer **Endlosschleife**.
 - **Beispiel:** in einer `while`-Schleife wird die Schleifenvariable nicht geändert!

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Das Programm wartet auf eine Eingabe (→ kein Fehler, Eingabe machen).
 - Es wartet auf Daten aus anderer Quelle (ggfs. Timeout vorsehen).
 - Es befindet sich in einer **Endlosschleife**.
 - **Beispiel:** in einer `while`-Schleife wird die Schleifenvariable nicht geändert!
- **Abbrechen** mit Ctrl-C oder *Restart Shell* in IDLE.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Das Programm wartet auf eine Eingabe (→ kein Fehler, Eingabe machen).
 - Es wartet auf Daten aus anderer Quelle (ggfs. Timeout vorsehen).
 - Es befindet sich in einer **Endlosschleife**.
 - **Beispiel:** in einer `while`-Schleife wird die Schleifenvariable nicht geändert!
- **Abbrechen** mit Ctrl-C oder *Restart Shell* in IDLE.
- Dann Fehler einkreisen und identifizieren (siehe **Debugging**)

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



■ Typische Fehler:

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Typische Fehler:
 - NameError: Benutzung einer nicht initialisierten Variablen.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



■ Typische Fehler:

- `NameError`: Benutzung einer nicht initialisierten Variablen.
- `TypeError`: Anderer Typ erwartet als dann tatsächlich benutzt wird.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



■ Typische Fehler:

- `NameError`: Benutzung einer nicht initialisierten Variablen.
- `TypeError`: Anderer Typ erwartet als dann tatsächlich benutzt wird.
- `IndexError`: Zugriff auf Sequenz über einen Index, der zu klein oder zu groß ist.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



■ Typische Fehler:

- `NameError`: Benutzung einer nicht initialisierten Variablen.
- `TypeError`: Anderer Typ erwartet als dann tatsächlich benutzt wird.
- `IndexError`: Zugriff auf Sequenz über einen Index, der zu klein oder zu groß ist.
- `AttributeError`: Versuch ein nicht existentes Attribut anzusprechen.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



■ Typische Fehler:

- `NameError`: Benutzung einer nicht initialisierten Variablen.
- `TypeError`: Anderer Typ erwartet als dann tatsächlich benutzt wird.
- `IndexError`: Zugriff auf Sequenz über einen Index, der zu klein oder zu groß ist.
- `AttributeError`: Versuch ein nicht existentes Attribut anzusprechen.
 - **Beispiel**: Zugriff auf Attribut `rigt`

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Typische Fehler:
 - `NameError`: Benutzung einer nicht initialisierten Variablen.
 - `TypeError`: Anderer Typ erwartet als dann tatsächlich benutzt wird.
 - `IndexError`: Zugriff auf Sequenz über einen Index, der zu klein oder zu groß ist.
 - `AttributeError`: Versuch ein nicht existentes Attribut anzusprechen.
 - **Beispiel**: Zugriff auf Attribut `rigt`
- Es gibt einen **Stack-Backtrace** mit genauer Angabe der Stelle.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Fehler tritt bei einer bestimmten Eingabe auf, aber die Ursache ist unklar.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Fehler tritt bei einer bestimmten Eingabe auf, aber die Ursache ist unklar.
- Suche **kleinere Eingabe**, bei der Fehler ebenfalls auftritt.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Fehler tritt bei einer bestimmten Eingabe auf, aber die Ursache ist unklar.
- Suche **kleinere Eingabe**, bei der Fehler ebenfalls auftritt.
 - **Beispiel:** Die Eingabe ist ein Baum

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Fehler tritt bei einer bestimmten Eingabe auf, aber die Ursache ist unklar.
- Suche **kleinere Eingabe**, bei der Fehler ebenfalls auftritt.
 - **Beispiel:** Die Eingabe ist ein Baum
 - Tritt der Fehler bereits bei einem Teilbaum auf?

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Fehler tritt bei einer bestimmten Eingabe auf, aber die Ursache ist unklar.
- Suche **kleinere Eingabe**, bei der Fehler ebenfalls auftritt.
 - **Beispiel:** Die Eingabe ist ein Baum
 - Tritt der Fehler bereits bei einem Teilbaum auf?
 - Liegt es an der Markierung der Wurzel?

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Fehler tritt bei einer bestimmten Eingabe auf, aber die Ursache ist unklar.
- Suche **kleinere Eingabe**, bei der Fehler ebenfalls auftritt.
 - **Beispiel:** Die Eingabe ist ein Baum
 - Tritt der Fehler bereits bei einem Teilbaum auf?
 - Liegt es an der Markierung der Wurzel?
 - Schneide Teilbäume (auf sinnvolle Art) ab um kleinere Eingaben zu erhalten.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Der Fehler tritt bei einer bestimmten Eingabe auf, aber die Ursache ist unklar.
- Suche **kleinere Eingabe**, bei der Fehler ebenfalls auftritt.
 - **Beispiel:** Die Eingabe ist ein Baum
 - Tritt der Fehler bereits bei einem Teilbaum auf?
 - Liegt es an der Markierung der Wurzel?
 - Schneide Teilbäume (auf sinnvolle Art) ab um kleinere Eingaben zu erhalten.
- Erstelle einen **Testfall** aus der Eingabe und der erwarteten Ausgabe (s.u.).

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Führe den so erstellten Testfall aus.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Führe den so erstellten Testfall aus.
- Das Ergebnis sollte fehlerhaft, d.h. anders als die erwartete Ausgabe, sein.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Führe den so erstellten Testfall aus.
- Das Ergebnis sollte fehlerhaft, d.h. anders als die erwartete Ausgabe, sein.
- **Testen ist die Ausführung eines Programms mit der Erwartung, dass es fehlschlägt.**

Programm-
entwicklung

Fehlertypen

Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Führe den so erstellten Testfall aus.
- Das Ergebnis sollte fehlerhaft, d.h. anders als die erwartete Ausgabe, sein.
- **Testen ist die Ausführung eines Programms mit der Erwartung, dass es fehlschlägt.**
- Gehe von der Fehlerstelle schrittweise **rückwärts** bis alles (Inhalte von Variablen und Attributen) wieder richtig erscheint.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Führe den so erstellten Testfall aus.
 - Das Ergebnis sollte fehlerhaft, d.h. anders als die erwartete Ausgabe, sein.
 - **Testen ist die Ausführung eines Programms mit der Erwartung, dass es fehlschlägt.**
 - Gehe von der Fehlerstelle schrittweise **rückwärts** bis alles (Inhalte von Variablen und Attributen) wieder richtig erscheint.
- ⇒ Der Fehler wurde durch die letzte Anweisung manifestiert.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Führe den so erstellten Testfall aus.
 - Das Ergebnis sollte fehlerhaft, d.h. anders als die erwartete Ausgabe, sein.
 - **Testen ist die Ausführung eines Programms mit der Erwartung, dass es fehlschlägt.**
 - Gehe von der Fehlerstelle schrittweise **rückwärts** bis alles (Inhalte von Variablen und Attributen) wieder richtig erscheint.
- ⇒ Der Fehler wurde durch die letzte Anweisung manifestiert.
- Enthält die Anweisung selbst einen Fehler?

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Führe den so erstellten Testfall aus.
 - Das Ergebnis sollte fehlerhaft, d.h. anders als die erwartete Ausgabe, sein.
 - **Testen ist die Ausführung eines Programms mit der Erwartung, dass es fehlschlägt.**
 - Gehe von der Fehlerstelle schrittweise **rückwärts** bis alles (Inhalte von Variablen und Attributen) wieder richtig erscheint.
- ⇒ Der Fehler wurde durch die letzte Anweisung manifestiert.
- Enthält die Anweisung selbst einen Fehler?
 - Falls nicht: Warum wurde sie ausgeführt? Das kann an umschließenden (fehlerhaften) bedingten Anweisungen liegen!

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Start mit dem Beispielausdruck

```
>>> e = Node('*', Node('+', leaf(2), leaf(5)), leaf(6))  
>>> print(expreval(e))
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<stdin>", line 7, in expreval

File "<stdin>", line 3, in expreval

NameError: name 'exprval' is not defined

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Start mit dem Beispielausdruck

```
>>> e = Node('*', Node('+', leaf(2), leaf(5)), leaf(6))
>>> print(expreval(e))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 7, in expreval
  File "<stdin>", line 3, in expreval
NameError: name 'exprval' is not defined
```

Verkleinern! Probiere linken Teilbaum

```
>>> print(expreval(e.left))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in expreval
NameError: name 'exprval' is not defined
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Weiter verkleinern! Probiere linken Teilbaum vom linken Teilbaum

```
>>> print(e.left.left)
Node(2, None, None)
>>> print(expreval(e.left.left))
None
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Weiter verkleinern! Probiere linken Teilbaum vom linken Teilbaum

```
>>> print(e.left.left)
Node(2, None, None)
>>> print(expreval(e.left.left))
None
```

- Hoppla, ein anderer Fehler!

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Weiter verkleinern! Probiere linken Teilbaum vom linken Teilbaum

```
>>> print(e.left.left)
Node(2, None, None)
>>> print(expreval(e.left.left))
None
```

- Hoppla, **ein anderer Fehler!**
- Offenbar wird der Fall, dass der Baum ein Blatt ist, nicht korrekt behandelt!

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Weiter verkleinern! Probiere linken Teilbaum vom linken Teilbaum

```
>>> print(e.left.left)
Node(2, None, None)
>>> print(expreval(e.left.left))
None
```

- Hoppla, **ein anderer Fehler!**
- Offenbar wird der Fall, dass der Baum ein Blatt ist, nicht korrekt behandelt!
- Abhilfe: Einfügen von `return tree.mark` am Ende.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Weiter verkleinern! Probiere linken Teilbaum vom linken Teilbaum

```
>>> print(e.left.left)
Node(2, None, None)
>>> print(expreval(e.left.left))
None
```

- Hoppla, **ein anderer Fehler!**
- Offenbar wird der Fall, dass der Baum ein Blatt ist, nicht korrekt behandelt!
- Abhilfe: Einfügen von `return tree.mark` am Ende.

Nach der Korrektur

```
>>> print(e.left.left)
Node(2, None, None)
>>> print(expreval(e.left.left))
2
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zurück zum linken Teilbaum

```
>>> print(expreval(e.left))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in expreval
NameError: name 'exprval' is not defined
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zurück zum linken Teilbaum

```
>>> print(expreval(e.left))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in expreval
NameError: name 'exprval' is not defined
```

- Kleinstes Beispiel, das den Fehler verursacht.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zurück zum linken Teilbaum

```
>>> print(expreval(e.left))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in expreval
NameError: name 'exprval' is not defined
```

- Kleinstes Beispiel, das den Fehler verursacht.
- `e.left` beginnt mit '+', also muss dort der Fehler sein.

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zurück zum linken Teilbaum

```
>>> print(expreval(e.left))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in expreval
NameError: name 'exprval' is not defined
```

- Kleinstes Beispiel, das den Fehler verursacht.
- `e.left` beginnt mit '+', also muss dort der Fehler sein.
- Korrigiere dort `exprval` nach `expreval`

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zurück zum linken Teilbaum

```
>>> print(exprval(e.left))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in exprval
NameError: name 'exprval' is not defined
```

- Kleinstes Beispiel, das den Fehler verursacht.
- `e.left` beginnt mit '+', also muss dort der Fehler sein.
- Korrigiere dort `exprval` nach `exprval`

Nach der Korrektur

```
>>> print(e.left)
Node('+', Node(2, None, None), Node(5, None, None))
>>> print(exprval(e.left))
7
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zurück zum kompletten Beispiel

```
>>> print (expreval (e))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 7, in expreval
AttributeError: 'Node' object has no attribute 'right'
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zurück zum kompletten Beispiel

```
>>> print (expreval (e))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 7, in expreval
AttributeError: 'Node' object has no attribute 'right'
```

- Keiner der Teilbäume liefert noch einen Fehler

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zurück zum kompletten Beispiel

```
>>> print (expreval (e))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 7, in expreval
AttributeError: 'Node' object has no attribute 'right'
```

- Keiner der Teilbäume liefert noch einen Fehler
- Problem muss an der Wurzel beim Operator '*' liegen

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zurück zum kompletten Beispiel

```
>>> print (expreval (e))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 7, in expreval
AttributeError: 'Node' object has no attribute 'rigt'
```

- Keiner der Teilbäume liefert noch einen Fehler
- Problem muss an der Wurzel beim Operator '*' liegen
- Korrigiere dort `rigt` nach `right`

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zurück zum kompletten Beispiel

```
>>> print (expreval (e))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 7, in expreval
AttributeError: 'Node' object has no attribute 'rigt'
```

- Keiner der Teilbäume liefert noch einen Fehler
- Problem muss an der Wurzel beim Operator '*' liegen
- Korrigiere dort rigt nach right

Nach der Korrektur

```
>>> print (expreval (e))
42
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Unser Programm enthielt 3 Fehler, die zu **Exceptions** führen.

Evaluating an Expression Tree

```
def exprval(tree : Node) -> int:
  if tree.mark == '+':
    return exprval(tree.left)+exprval(tree.right)
  elif tree.mark == '-':
    return exprval(tree.left)-exprval(tree.right)
  elif tree.mark == '*':
    return exprval(tree.left)*exprval(tree.right)
  elif tree.mark == '/':
    return exprval(tree.left)/exprval(tree.right)
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Unser Programm enthielt 3 Fehler, die zu **Exceptions** führen.
- Das korrekte Programm:

Evaluating an Expression Tree

```
def expreval(tree : Node) -> int:  
  if tree.mark == '+':  
    return expreval(tree.left)+expreval(tree.right)  
  elif tree.mark == '-':  
    return expreval(tree.left)-expreval(tree.right)  
  elif tree.mark == '*':  
    return expreval(tree.left)*expreval(tree.right)  
  elif tree.mark == '/':  
    return expreval(tree.left)/expreval(tree.right)  
  else:  
    return tree.mark
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler
Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Ein logischer Fehler liegt vor, wenn das Verhalten/die Ausgabe des Programms vom **erwarteten** Verhalten abweicht.

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Ein logischer Fehler liegt vor, wenn das Verhalten/die Ausgabe des Programms vom **erwarteten** Verhalten abweicht.
 - **Beispiele:** Statt Addition wird eine Multiplikation durchgeführt, metrische und imperiale Werte werden ohne Konversion verglichen.

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Ein logischer Fehler liegt vor, wenn das Verhalten/die Ausgabe des Programms vom **erwarteten** Verhalten abweicht.
 - **Beispiele:** Statt Addition wird eine Multiplikation durchgeführt, metrische und imperiale Werte werden ohne Konversion verglichen.
- Tatsächlich liegt hier erst dann ein Fehler vor, wenn das erwartete Verhalten (**formal**) **spezifiziert** war. Aber auch informelle Vorgaben können natürlich verletzt werden.

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:

Fehlerfreies
Programmieren?

Zusammen-
fassung



- Ein logischer Fehler liegt vor, wenn das Verhalten/die Ausgabe des Programms vom **erwarteten** Verhalten abweicht.
 - **Beispiele:** Statt Addition wird eine Multiplikation durchgeführt, metrische und imperiale Werte werden ohne Konversion verglichen.
- Tatsächlich liegt hier erst dann ein Fehler vor, wenn das erwartete Verhalten **(formal) spezifiziert** war. Aber auch informelle Vorgaben können natürlich verletzt werden.
- Im industriellen Umfeld wird dafür ein **Pflichtenheft** erstellt.

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Ein logischer Fehler liegt vor, wenn das Verhalten/die Ausgabe des Programms vom **erwarteten** Verhalten abweicht.
 - **Beispiele:** Statt Addition wird eine Multiplikation durchgeführt, metrische und imperiale Werte werden ohne Konversion verglichen.
- Tatsächlich liegt hier erst dann ein Fehler vor, wenn das erwartete Verhalten (**formal**) **spezifiziert** war. Aber auch informelle Vorgaben können natürlich verletzt werden.
- Im industriellen Umfeld wird dafür ein **Pflichtenheft** erstellt.
- **Best practice:** Beschreibe das erwartete Verhalten durch **Beispiele** / Testfälle.

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:

Fehlerfreies

Programmieren?

Zusammen-

fassung



- Ein logischer Fehler liegt vor, wenn das Verhalten/die Ausgabe des Programms vom **erwarteten** Verhalten abweicht.
 - **Beispiele:** Statt Addition wird eine Multiplikation durchgeführt, metrische und imperiale Werte werden ohne Konversion verglichen.
- Tatsächlich liegt hier erst dann ein Fehler vor, wenn das erwartete Verhalten (**formal**) **spezifiziert** war. Aber auch informelle Vorgaben können natürlich verletzt werden.
- Im industriellen Umfeld wird dafür ein **Pflichtenheft** erstellt.
- **Best practice:** Beschreibe das erwartete Verhalten durch **Beispiele** / Testfälle.
- Unvollständig, aber besser als nichts!

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:

Fehlerfreies

Programmieren?

Zusammen-

fassung



- Gibt es logische Fehler in unserem Programm?
- Wir hatten ganzzahlige Arithmetik gefordert, aber der Operator „/“ liefert eine Gleitkommazahl!

Evaluating an Expression Tree

```
def expreval(tree):  
    if tree.mark == '+':  
        return expreval(tree.left)+expreval(tree.right)  
    elif tree.mark == '-':  
        return expreval(tree.left)-expreval(tree.right)  
    elif tree.mark == '*':  
        return expreval(tree.left)*expreval(tree.right)  
    elif tree.mark == '/':  
        return expreval(tree.left)/expreval(tree.right)  
    else:  
        return tree.mark
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Gibt es logische Fehler in unserem Programm?
- Wir hatten ganzzahlige Arithmetik gefordert, aber der Operator „/“ liefert eine Gleitkommazahl!

Evaluating an Expression Tree

```
def expreval(tree):  
    if tree.mark == '+':  
        return expreval(tree.left)+expreval(tree.right)  
    elif tree.mark == '-':  
        return expreval(tree.left)-expreval(tree.right)  
    elif tree.mark == '*':  
        return expreval(tree.left)*expreval(tree.right)  
    elif tree.mark == '/':  
        return expreval(tree.left)//expreval(tree.right)  
    else:  
        return tree.mark
```

Programm-
entwicklung

Fehlertypen
Syntaktische
Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Welche Beispiele hätten wir gebraucht?

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Welche Beispiele hätten wir gebraucht?

- Ein Beispiel für einen konstanten Ausdruck.

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Welche Beispiele hätten wir gebraucht?

- Ein Beispiel für einen konstanten Ausdruck.
- Je ein Beispiel pro Operator.

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Welche Beispiele hätten wir gebraucht?

- Ein Beispiel für einen konstanten Ausdruck.
- Je ein Beispiel pro Operator.
- Für die Division ein Beispiel, bei dem ganzzahlig dividiert werden muss.
Z.B. Node ('/', leaf(5), leaf(3))

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Welche Beispiele hätten wir gebraucht?

- Ein Beispiel für einen konstanten Ausdruck.
- Je ein Beispiel pro Operator.
- Für die Division ein Beispiel, bei dem ganzzahlig dividiert werden muss.
Z.B. Node ('/', leaf(5), leaf(3))

Diese Beispiele hätten alle Fehler identifiziert!

- Für jede bedingte Anweisung im Programm gibt es einen Test, der die Bedingung wahr macht.
- Für jede Anweisung im Programm gibt es einen Test, der zu ihrer Ausführung führt.
- Randfälle und Sonderfälle müssen abgedeckt werden.

Programm-
entwicklung

Fehlertypen

Syntaktische

Fehler

Laufzeitfehler

Logische Fehler

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?
?

Zusammen-
fassung



Debuggen

Programm-
entwicklung

Debuggen

Print-Anweisungen

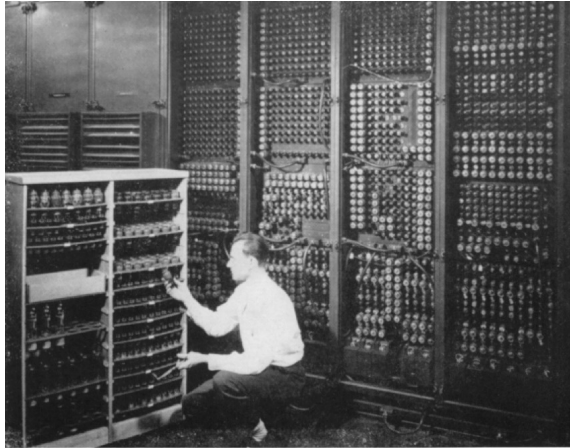
Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung




9/9

0800 Antam started
 1000 " stopped - antam ✓ { 1.2700 9.037 847 025
 1300 (032) MP-MC ~~1.452 449 000~~ 9.037 846 995 correct
 (033) PRO 2 2.130 476 415 (2) 4.615 925 059 (-2)
 correct 2.130 476 415
 2.130 676 415

Relays 6-2 in 033 failed special speed test
 in relay " 11,00 test.

Relays changed

1100 Started Cosine Tape (Sine check)
 1525 Started Multi-Adder Test.

1545  Relay #70 Panel F
 (moth) in relay.

First actual case of bug being found.

1650 antam started.
 1700 closed down.

Relay 2145
 Relay 2370

- Programm-entwicklung
- Debuggen
 - Print-Anweisungen
 - Debugger
 - Debugging-Techniken
- Tests
- Ausblick: Fehlerfreies Programmieren?
- Zusammenfassung

Debuggen = Käfer jagen und töten



- In den frühen Computern haben Motten/Fliegen/Käfer (engl. *Bug*) durch Kurzschlüsse für Fehlfunktionen gesorgt.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Debuggen = Käfer jagen und töten



- In den frühen Computern haben Motten/Fliegen/Käfer (engl. *Bug*) durch Kurzschlüsse für Fehlfunktionen gesorgt.
- Diese Käfer (oder andere Ursachen für Fehlfunktionen) zu finden heißt *debuggen*, im Deutschen manchmal *entwanzen*.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Debuggen = Käfer jagen und töten



- In den frühen Computern haben Motten/Fliegen/Käfer (engl. *Bug*) durch Kurzschlüsse für Fehlfunktionen gesorgt.
- Diese Käfer (oder andere Ursachen für Fehlfunktionen) zu finden heißt *debuggen*, im Deutschen manchmal *entwanzen*.
- Hat viel von **Detektivarbeit** (wer ist der Schuldige?)

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Debuggen = Käfer jagen und töten



- In den frühen Computern haben Motten/Fliegen/Käfer (engl. *Bug*) durch Kurzschlüsse für Fehlfunktionen gesorgt.
- Diese Käfer (oder andere Ursachen für Fehlfunktionen) zu finden heißt *debuggen*, im Deutschen manchmal *entwanzen*.
- Hat viel von **Detektivarbeit** (wer ist der Schuldige?)
- Aber nicht mystifizieren; vieles ist heute systematisiert und automatisierbar.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Debuggen = Käfer jagen und töten



- In den frühen Computern haben Motten/Fliegen/Käfer (engl. *Bug*) durch Kurzschlüsse für Fehlfunktionen gesorgt.
- Diese Käfer (oder andere Ursachen für Fehlfunktionen) zu finden heißt *debuggen*, im Deutschen manchmal *entwanzen*.
- Hat viel von **Detektivarbeit** (wer ist der Schuldige?)
- Aber nicht mystifizieren; vieles ist heute systematisiert und automatisierbar.
- Die Verbesserungen heißen **Bugfixes** – und sollten das Problem dann lösen!

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung

Debuggen = Käfer jagen und töten



- In den frühen Computern haben Motten/Fliegen/Käfer (engl. *Bug*) durch Kurzschlüsse für Fehlfunktionen gesorgt.
- Diese Käfer (oder andere Ursachen für Fehlfunktionen) zu finden heißt *debuggen*, im Deutschen manchmal *entwanzen*.
- Hat viel von **Detektivarbeit** (wer ist der Schuldige?)
- Aber nicht mystifizieren; vieles ist heute systematisiert und automatisierbar.
- Die Verbesserungen heißen **Bugfixes** – und sollten das Problem dann lösen!
- Oft werden durch Bugfixes neue Fehler erzeugt. . .

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Most coders think debugging software is about fixing a mistake, but that is bullshit.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Most coders think debugging software is about fixing a mistake, but that is bullshit. Debugging is actually all about finding the bug, about understanding why the bug was there to begin with, about knowing that its existence was no accident.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Most coders think debugging software is about fixing a mistake, but that is bullshit. Debugging is actually all about finding the bug, about understanding why the bug was there to begin with, about knowing that its existence was no accident. It came to you to deliver a message, like an unconscious bubble floating to the surface, popping with a revelation you've secretly known all along.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



*Most coders think debugging software is about fixing a mistake, but that is bullshit. Debugging is actually all about finding the bug, **about understanding why the bug was there to begin with, about knowing that its existence was no accident.** It came to you to deliver a message, like an unconscious bubble floating to the surface, popping with a revelation you've secretly known all along.*

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:

Fehlerfreies

Programmieren?

Zusammen-
fassung



Das Wichtigste

Versuche minimale Eingaben zu finden, die den Fehler hervorrufen!

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Das Wichtigste

Versuche minimale Eingaben zu finden, die den Fehler hervorrufen!

Nachvollziehen der Berechnung bis zum Fehler

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Das Wichtigste

Versuche minimale Eingaben zu finden, die den Fehler hervorrufen!

Nachvollziehen der Berechnung bis zum Fehler

- 1 Kleine Beispiele von Hand oder mit `pythontutor`

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Das Wichtigste

Versuche minimale Eingaben zu finden, die den Fehler hervorrufen!

Nachvollziehen der Berechnung bis zum Fehler

- 1 Kleine Beispiele von Hand oder mit `pythontutor`
- 2 Falls kein Debugger verfügbar: Modifikation des Programms zur Ausgabe von bestimmten Variablenwerten an bestimmten Stellen (Einfügen von `print`-Anweisungen)

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Das Wichtigste

Versuche minimale Eingaben zu finden, die den Fehler hervorrufen!

Nachvollziehen der Berechnung bis zum Fehler

- 1 Kleine Beispiele von Hand oder mit `pythontutor`
- 2 Falls kein Debugger verfügbar: Modifikation des Programms zur Ausgabe von bestimmten Variablenwerten an bestimmten Stellen (Einfügen von `print`-Anweisungen)
- 3 Einsatz von Debugging-Werkzeugen: **Post-Mortem-Analyse-Tools** und **Debugger**

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Beobachten von internen Werten (vgl. bei Hardware mit einem Oszilloskop)

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Beobachten von internen Werten (vgl. bei Hardware mit einem Oszilloskop)
- In vielen Sprachen/Systemen können `print`-Anweisungen eingefügt werden.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Beobachten von internen Werten (vgl. bei Hardware mit einem Oszilloskop)
- In vielen Sprachen/Systemen können `print`-Anweisungen eingefügt werden.
- Einfachste Möglichkeit das Verhalten eines Programmes zu beobachten.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Beobachten von internen Werten (vgl. bei Hardware mit einem Oszilloskop)
- In vielen Sprachen/Systemen können `print`-Anweisungen eingefügt werden.
- Einfachste Möglichkeit das Verhalten eines Programmes zu beobachten.
 - **Achtung:** Zusätzliche Ausgaben können das Verhalten (speziell das Zeitverhalten) signifikant ändern!

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Beobachten von internen Werten (vgl. bei Hardware mit einem Oszilloskop)
- In vielen Sprachen/Systemen können `print`-Anweisungen eingefügt werden.
- Einfachste Möglichkeit das Verhalten eines Programmes zu beobachten.
 - **Achtung:** Zusätzliche Ausgaben können das Verhalten (speziell das Zeitverhalten) signifikant ändern!
- Eine generalisierte Form ist das *Logging*, bei dem `prints` generell im Code integriert sind und mit Schaltern an- und abgestellt werden können.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:

Fehlerfreies

Programmieren?

Zusammen-

fassung



1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 *Post-Mortem-Tools*: Analyse des Programmszustands nach einem Fehler
 - Stack Backtrace wie in Python

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler
 - Stack Backtrace wie in Python
 - Früher: Speicherbelegung (Hex-Dump)

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler
 - Stack Backtrace wie in Python
 - Früher: Speicherbelegung (Hex-Dump)
 - Heute: Variablenbelegung (global und lokal in der Kellertabelle)

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler
 - Stack Backtrace wie in Python
 - Früher: Speicherbelegung (Hex-Dump)
 - Heute: Variablenbelegung (global und lokal in der Kellertabelle)
- 2 *Interaktive Debugger*

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler
 - Stack Backtrace wie in Python
 - Früher: Speicherbelegung (Hex-Dump)
 - Heute: Variablenbelegung (global und lokal in der Kellertabelle)
- 2 *Interaktive Debugger*
 - Setzen von Breakpoints (u.U. konditional)

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler

- Stack Backtrace wie in Python
- Früher: Speicherbelegung (Hex-Dump)
- Heute: Variablenbelegung (global und lokal in der Kellertabelle)

2 *Interaktive Debugger*

- Setzen von Breakpoints (u.U. konditional)
- Inspektion des Programmzustands (Variablenbelegung)

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler

- Stack Backtrace wie in Python
- Früher: Speicherbelegung (Hex-Dump)
- Heute: Variablenbelegung (global und lokal in der Kellertabelle)

2 *Interaktive Debugger*

- Setzen von Breakpoints (u.U. konditional)
- Inspektion des Programmzustands (Variablenbelegung)
- Ändern des Zustands

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler

- Stack Backtrace wie in Python
- Früher: Speicherbelegung (Hex-Dump)
- Heute: Variablenbelegung (global und lokal in der Kellertabelle)

2 *Interaktive Debugger*

- Setzen von Breakpoints (u.U. konditional)
- Inspektion des Programmzustands (Variablenbelegung)
- Ändern des Zustands
- Einzelschrittausführung (Stepping / Tracing):

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler

- Stack Backtrace wie in Python
- Früher: Speicherbelegung (Hex-Dump)
- Heute: Variablenbelegung (global und lokal in der Kellertabelle)

2 *Interaktive Debugger*

- Setzen von Breakpoints (u.U. konditional)
- Inspektion des Programmzustands (Variablenbelegung)
- Ändern des Zustands
- Einzelschrittausführung (Stepping / Tracing):

Step in: Mache einen Schritt, ggfs. in eine Funktion hinein

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler

- Stack Backtrace wie in Python
- Früher: Speicherbelegung (Hex-Dump)
- Heute: Variablenbelegung (global und lokal in der Kellertabelle)

2 *Interaktive Debugger*

- Setzen von Breakpoints (u.U. konditional)
- Inspektion des Programmzustands (Variablenbelegung)
- Ändern des Zustands
- Einzelschrittausführung (Stepping / Tracing):

Step in: Mache einen Schritt, ggfs. in eine Funktion hinein

Step over: Mache einen Schritt, führe dabei ggfs. eine Funktion aus

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler

- Stack Backtrace wie in Python
- Früher: Speicherbelegung (Hex-Dump)
- Heute: Variablenbelegung (global und lokal in der Kellertabelle)

2 *Interaktive Debugger*

- Setzen von Breakpoints (u.U. konditional)
- Inspektion des Programmzustands (Variablenbelegung)
- Ändern des Zustands
- Einzelschrittausführung (Stepping / Tracing):

Step in: Mache einen Schritt, ggfs. in eine Funktion hinein

Step over: Mache einen Schritt, führe dabei ggfs. eine Funktion aus

Step out: Beende den aktuellen Funktionsaufruf

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler

- Stack Backtrace wie in Python
- Früher: Speicherbelegung (Hex-Dump)
- Heute: Variablenbelegung (global und lokal in der Kellertabelle)

2 *Interaktive Debugger*

- Setzen von Breakpoints (u.U. konditional)
- Inspektion des Programmzustands (Variablenbelegung)
- Ändern des Zustands
- Einzelschrittausführung (Stepping / Tracing):

Step in: Mache einen Schritt, ggfs. in eine Funktion hinein

Step over: Mache einen Schritt, führe dabei ggfs. eine Funktion aus

Step out: Beende den aktuellen Funktionsaufruf

Go/Continue: Starte Ausführung bzw. setze fort

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



1 *Post-Mortem-Tools*: Analyse des Programmzustands nach einem Fehler

- Stack Backtrace wie in Python
- Früher: Speicherbelegung (Hex-Dump)
- Heute: Variablenbelegung (global und lokal in der Kellertabelle)

2 *Interaktive Debugger*

- Setzen von Breakpoints (u.U. konditional)
- Inspektion des Programmzustands (Variablenbelegung)
- Ändern des Zustands
- Einzelschrittausführung (Stepping / Tracing):

Step in: Mache einen Schritt, ggfs. in eine Funktion hinein

Step over: Mache einen Schritt, führe dabei ggfs. eine Funktion aus

Step out: Beende den aktuellen Funktionsaufruf

Go/Continue: Starte Ausführung bzw. setze fort

Quit: Beendet alles.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 pdb ist ein Konsolen-orientierter Debugger, der auch Post-Mortem-Analyse anbietet (siehe <http://docs.python.org/3/library/pdb.html>).

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 pdb ist ein Konsolen-orientierter Debugger, der auch Post-Mortem-Analyse anbietet (siehe <http://docs.python.org/3/library/pdb.html>).
- 2 IDLE enthält einen weniger mächtigen, aber einfach zu bedienenden GUI-Debugger. Im Debug-Menü:

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 pdb ist ein Konsolen-orientierter Debugger, der auch Post-Mortem-Analyse anbietet (siehe <http://docs.python.org/3/library/pdb.html>).
- 2 IDLE enthält einen weniger mächtigen, aber einfach zu bedienenden GUI-Debugger. Im Debug-Menü:
 - *Goto File/Line*: Wenn der Cursor in einer Traceback-Zeile steht, springt der Editor zur angegebenen Stelle.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 `pdb` ist ein Konsolen-orientierter Debugger, der auch Post-Mortem-Analyse anbietet (siehe <http://docs.python.org/3/library/pdb.html>).
- 2 IDLE enthält einen weniger mächtigen, aber einfach zu bedienenden GUI-Debugger. Im Debug-Menü:
 - *Goto File/Line*: Wenn der Cursor in einer Traceback-Zeile steht, springt der Editor zur angegebenen Stelle.
 - *Stack Viewer*: Erlaubt eine Post-Mortem-Analyse des letzten durch eine Exception beendeten Programmlaufs.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 `pdb` ist ein Konsolen-orientierter Debugger, der auch Post-Mortem-Analyse anbietet (siehe <http://docs.python.org/3/library/pdb.html>).
- 2 IDLE enthält einen weniger mächtigen, aber einfach zu bedienenden GUI-Debugger. Im Debug-Menü:
 - *Goto File/Line*: Wenn der Cursor in einer Traceback-Zeile steht, springt der Editor zur angegebenen Stelle.
 - *Stack Viewer*: Erlaubt eine Post-Mortem-Analyse des letzten durch eine Exception beendeten Programmlaufs.
 - *Debugger*: Startet den Debug-Modus:

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 pdb ist ein Konsolen-orientierter Debugger, der auch Post-Mortem-Analyse anbietet (siehe <http://docs.python.org/3/library/pdb.html>).
- 2 IDLE enthält einen weniger mächtigen, aber einfach zu bedienenden GUI-Debugger. Im Debug-Menü:
 - *Goto File/Line*: Wenn der Cursor in einer Traceback-Zeile steht, springt der Editor zur angegebenen Stelle.
 - *Stack Viewer*: Erlaubt eine Post-Mortem-Analyse des letzten durch eine Exception beendeten Programmlaufs.
 - *Debugger*: Startet den Debug-Modus:
 - Es erscheint ein Fenster, in dem der Aufruf-Stapel, globale und lokale Variablen angezeigt werden. Ggfs. wird auch der aktuelle Quellcode angezeigt.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 `pdb` ist ein Konsolen-orientierter Debugger, der auch Post-Mortem-Analyse anbietet (siehe <http://docs.python.org/3/library/pdb.html>).
- 2 IDLE enthält einen weniger mächtigen, aber einfach zu bedienenden GUI-Debugger. Im Debug-Menü:
 - *Goto File/Line*: Wenn der Cursor in einer Traceback-Zeile steht, springt der Editor zur angegebenen Stelle.
 - *Stack Viewer*: Erlaubt eine Post-Mortem-Analyse des letzten durch eine Exception beendeten Programmlaufs.
 - *Debugger*: Startet den Debug-Modus:
 - Es erscheint ein Fenster, in dem der Aufruf-Stapel, globale und lokale Variablen angezeigt werden. Ggfs. wird auch der aktuelle Quellcode angezeigt.
 - Zum Setzen von Breakpoints, muss im Quellcode eine Zeile rechts-geklickt werden (Mac: Ctrl-Klick).

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 pdb ist ein Konsolen-orientierter Debugger, der auch Post-Mortem-Analyse anbietet (siehe <http://docs.python.org/3/library/pdb.html>).
- 2 IDLE enthält einen weniger mächtigen, aber einfach zu bedienenden GUI-Debugger. Im Debug-Menü:
 - *Goto File/Line*: Wenn der Cursor in einer Traceback-Zeile steht, springt der Editor zur angegebenen Stelle.
 - *Stack Viewer*: Erlaubt eine Post-Mortem-Analyse des letzten durch eine Exception beendeten Programmlaufs.
 - *Debugger*: Startet den Debug-Modus:
 - Es erscheint ein Fenster, in dem der Aufruf-Stapel, globale und lokale Variablen angezeigt werden. Ggfs. wird auch der aktuelle Quellcode angezeigt.
 - Zum Setzen von Breakpoints, muss im Quellcode eine Zeile rechts-geklickt werden (Mac: Ctrl-Klick).
 - Stepping mit den Go/Step usw. Knöpfen.

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 Formulare eine Hypothese, warum der Fehler auftritt, an welcher Stelle des Programms sich der Fehler manifestiert, welche Variablen betroffen sind!

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 Formulare eine Hypothese, warum der Fehler auftritt, an welcher Stelle des Programms sich der Fehler manifestiert, welche Variablen betroffen sind!
- 2 **Instrumentiere** die Stelle, sodass die betroffenen Variablen inspiziert werden können (Breakpoints oder `print`-Anweisungen)

Programm-
entwicklung

Debuggen

Print-Anweisungen

Debugger

Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 Formulare eine Hypothese, warum der Fehler auftritt, an welcher Stelle des Programms sich der Fehler manifestiert, welche Variablen betroffen sind!
- 2 **Instrumentiere** die Stelle, sodass die betroffenen Variablen inspiziert werden können (Breakpoints oder `print`-Anweisungen)
- 3 Versuche zu verstehen, wie es zu dem Fehler kommt: Was ist die tiefere Ursache?

Programm-
entwicklung

Debuggen
Print-Anweisungen
Debugger
Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 Formuliere eine Hypothese, warum der Fehler auftritt, an welcher Stelle des Programms sich der Fehler manifestiert, welche Variablen betroffen sind!
- 2 **Instrumentiere** die Stelle, sodass die betroffenen Variablen inspiziert werden können (Breakpoints oder `print`-Anweisungen)
- 3 Versuche zu verstehen, wie es zu dem Fehler kommt: Was ist die tiefere Ursache?
- 4 Formuliere einen **Bugfix** erst dann, wenn das **Problem verstanden** ist. Einfache Lösungen sind oft nicht hilfreich.

Programm-
entwicklung

Debuggen
Print-Anweisungen
Debugger
Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 Formuliere eine Hypothese, warum der Fehler auftritt, an welcher Stelle des Programms sich der Fehler manifestiert, welche Variablen betroffen sind!
- 2 **Instrumentiere** die Stelle, sodass die betroffenen Variablen inspiziert werden können (Breakpoints oder `print`-Anweisungen)
- 3 Versuche zu verstehen, wie es zu dem Fehler kommt: Was ist die tiefere Ursache?
- 4 Formuliere einen **Bugfix** erst dann, wenn das **Problem verstanden** ist. Einfache Lösungen sind oft nicht hilfreich.
- 5 Teste nach dem Bugfix, ob das Problem tatsächlich beseitigt wurde.

Programm-
entwicklung

Debuggen
Print-Anweisungen
Debugger
Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 Formuliere eine Hypothese, warum der Fehler auftritt, an welcher Stelle des Programms sich der Fehler manifestiert, welche Variablen betroffen sind!
- 2 **Instrumentiere** die Stelle, sodass die betroffenen Variablen inspiziert werden können (Breakpoints oder `print`-Anweisungen)
- 3 Versuche zu verstehen, wie es zu dem Fehler kommt: Was ist die tiefere Ursache?
- 4 Formuliere einen **Bugfix** erst dann, wenn das **Problem verstanden** ist. Einfache Lösungen sind oft nicht hilfreich.
- 5 Teste nach dem Bugfix, ob das Problem tatsächlich beseitigt wurde.
- 6 Weitere Tests laufen lassen (s.u.).

Programm-
entwicklung

Debuggen
Print-Anweisungen
Debugger
Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- 1 Formuliere eine Hypothese, warum der Fehler auftritt, an welcher Stelle des Programms sich der Fehler manifestiert, welche Variablen betroffen sind!
- 2 **Instrumentiere** die Stelle, sodass die betroffenen Variablen inspiziert werden können (Breakpoints oder `print`-Anweisungen)
- 3 Versuche zu verstehen, wie es zu dem Fehler kommt: Was ist die tiefere Ursache?
- 4 Formuliere einen **Bugfix** erst dann, wenn das **Problem verstanden** ist. Einfache Lösungen sind oft nicht hilfreich.
- 5 Teste nach dem Bugfix, ob das Problem tatsächlich beseitigt wurde.
- 6 Weitere Tests laufen lassen (s.u.).
- 7 Wenn es nicht weiter geht: frische Luft und eine Tasse Kaffee hilft!

Programm-
entwicklung

Debuggen
Print-Anweisungen
Debugger
Debugging-
Techniken

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Automatische Tests

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Testen eines Programms heißt, **fehlerhaftes Verhalten zu provozieren.**

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Testen eines Programms heißt, **fehlerhaftes Verhalten zu provozieren**.
- Ein **Testfall** besteht aus einer Eingabe und dem erwarteten Ergebnis. Die Testfälle bilden eine **Testsuite**.

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Testen eines Programms heißt, **fehlerhaftes Verhalten zu provozieren**.
- Ein **Testfall** besteht aus einer Eingabe und dem erwarteten Ergebnis. Die Testfälle bilden eine **Testsuite**.
- **Schon vor dem Programmieren** systematisch Testfälle in Form von Beispielen erstellen:

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Testen eines Programms heißt, **fehlerhaftes Verhalten zu provozieren**.
- Ein **Testfall** besteht aus einer Eingabe und dem erwarteten Ergebnis. Die Testfälle bilden eine **Testsuite**.
- **Schon vor dem Programmieren** systematisch Testfälle in Form von Beispielen erstellen:
 - Basisfälle und Grenzfälle (z.B. erstes bzw letztes Element in einer Datenstruktur)

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmie-
ren?

Zusammen-
fassung



- Testen eines Programms heißt, **fehlerhaftes Verhalten zu provozieren**.
- Ein **Testfall** besteht aus einer Eingabe und dem erwarteten Ergebnis. Die Testfälle bilden eine **Testsuite**.
- **Schon vor dem Programmieren** systematisch Testfälle in Form von Beispielen erstellen:
 - Basisfälle und Grenzfälle (z.B. erstes bzw letztes Element in einer Datenstruktur)
 - Jede Anweisung im Code soll durch einen Testfall abgedeckt (d.h. ausgeführt) werden (**Coverage**)

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Testen eines Programms heißt, **fehlerhaftes Verhalten zu provozieren**.
- Ein **Testfall** besteht aus einer Eingabe und dem erwarteten Ergebnis. Die Testfälle bilden eine **Testsuite**.
- **Schon vor dem Programmieren** systematisch Testfälle in Form von Beispielen erstellen:
 - Basisfälle und Grenzfälle (z.B. erstes bzw letztes Element in einer Datenstruktur)
 - Jede Anweisung im Code soll durch einen Testfall abgedeckt (d.h. ausgeführt) werden (**Coverage**)
 - Versuche Eingaben zu finden, die die Bedingungen im Programm unabhängig voneinander wahr bzw. falsch machen (soweit möglich).

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Testen eines Programms heißt, **fehlerhaftes Verhalten zu provozieren**.
- Ein **Testfall** besteht aus einer Eingabe und dem erwarteten Ergebnis. Die Testfälle bilden eine **Testsuite**.
- **Schon vor dem Programmieren** systematisch Testfälle in Form von Beispielen erstellen:
 - Basisfälle und Grenzfälle (z.B. erstes bzw letztes Element in einer Datenstruktur)
 - Jede Anweisung im Code soll durch einen Testfall abgedeckt (d.h. ausgeführt) werden (**Coverage**)
 - Versuche Eingaben zu finden, die die Bedingungen im Programm unabhängig voneinander wahr bzw. falsch machen (soweit möglich).
- **Beim Programmieren**: Tests, die zur Entdeckung eines Fehlers geführt haben, müssen aufbewahrt werden!

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Testgetriebene Entwicklung (test-driven development, TDD)

- Zu Beginn der Entwicklung eines Systems werden Testfälle (Anwendungsbeispiele) formuliert, die nach und nach erfüllt werden
- Der **Fortschritt der Entwicklung** des Systems kann dann mit Hilfe der Anzahl der bestandenen Tests gemessen werden.

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Testgetriebene Entwicklung (test-driven development, TDD)

- Zu Beginn der Entwicklung eines Systems werden Testfälle (Anwendungsbeispiele) formuliert, die nach und nach erfüllt werden
- Der **Fortschritt der Entwicklung** des Systems kann dann mit Hilfe der Anzahl der bestandenen Tests gemessen werden.

Regressionstest

- Durch Bugfix oder Erweiterung können neue Fehler eingeschleppt werden oder alte Fehler können erneut aufbrechen
- Der **Regressionstest** wiederholt nach jeder Änderung sämtliche Tests um sicher zu stellen, dass die bestehende Funktionalität erhalten bleibt.

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- **Unittests** sollen sicherstellen, dass die Einzelteile eines System funktionieren
- Sie enthalten Testfälle für Teile eines Systems (Modul, Funktion, usw.).
- Um Regressionstests und TDD zu ermöglichen, werden die Unittests automatisch ausgeführt.
- In Python gibt es eine Reihe von Werkzeugen zur Automatisierung.
- Eins davon ist `pytest`

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

`pytest`

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- `py.test` ist ein umfassendes Framework, um Tests zu schreiben
- Installation durch `pip3 install pytest`.
- Idee: Für jede FUT (**function under test**) werden eine oder mehrere **Testfunktionen** geschrieben. Der Name einer Testfunktion beginnt immer mit `test_`.
- In der Testfunktion werden die erwarteten Rückgabewerte der FUTs als **Assertions** mit der `assert`-Anweisung formuliert.
- `assert`-Anweisung:
`assert Bedingung [, String]`
- `assert` sichert zu, dass die *Bedingung* wahr ist. Wenn das nicht der Fall ist, wird eine **Exception** ausgelöst und der *String* ausgegeben.

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Beispiel für einen Testfall

```
import pytest
...
def test_expreval_b():
    """Test of expreval that fails."""
    expr = Node('*', Node('+', leaf(3), leaf(5)),
                 leaf(6))
    assert expreval(expr) == 42
```

- Ausführung mit `pytest expreval.py`

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Die Ausgabe in obigem Beispiel:

```
===== test session starts =====
expreval.py::test_expreval_b FAILED
===== FAILURES =====
----- test_expreval_b -----

    def test_expreval_b():
        """Test of expreval that fails."""
        expr = Node('*', Node('+', leaf(3), leaf(5)),
                    leaf(6))
>       assert expreval(expr) == 42
E       assert 48 == 42
E       + where 48 = expreval(Node('*', Node('+', leaf(3), leaf(5)), leaf(6)))

expreval.py:50: AssertionError
===== 1 failed, 1 passed in 0.02 seconds =====
```

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Testsuite wie oben besprochen

```
import pytest
...
def test_expreval_1():
    """Testing expreval from example."""
    e = Node('*', Node('+', leaf(2), leaf(5)), leaf(6))
    assert expreval (e.left.left) == 2
    assert expreval (e.left) == 7
    assert expreval (e) == 42

def test_expreval_2():
    """Testing logical bug in expreeval"""
    assert expreval (Node ('/', leaf(2), leaf(3))) == 0
```

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Ausgabe vor Bugfix des logischen Fehlers

```
[Peters-MacBook-Pro:python thiemann$ pytest trees.py ]
===== test session starts =====
platform darwin -- Python 3.7.0, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: /Users/thiemann/svn/teaching/info2018/slides/python, inifile:
collected 2 items

trees.py .F [100%]

===== FAILURES =====
----- test_expreval_2 -----

    def test_expreval_2():
>     assert expreval (Node ('/', leaf(2), leaf(3))) == 0
E       AssertionError: assert 0.6666666666666666 == 0
E         + where 0.6666666666666666 = expreval(<trees.Node object at 0x10f4ed978>)
E         + where <trees.Node object at 0x10f4ed978> = Node('/', <trees.Node object at 0x10f4ede48>, <trees.Node object at 0x10f4ed898>)
E           + where <trees.Node object at 0x10f4ede48> = leaf(2)
E           + and <trees.Node object at 0x10f4ed898> = leaf(3)

trees.py:35: AssertionError
===== 1 failed, 1 passed in 0.07 seconds =====
```

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Ausgabe nach Bugfix

```
[Peters-MacBook-Pro:python thiemann$ pytest trees.py ]
===== test session starts =====
platform darwin -- Python 3.7.0, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: /Users/thiemann/svn/teaching/info2018/slides/python, inifile:
collected 2 items

trees.py .. [100%]

===== 2 passed in 0.02 seconds =====
```

Programm-
entwicklung

Debuggen

Tests

Testgetriebene
Entwicklung

Unittests

pytest

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Ausblick: Fehlerfreies Programmieren?

Programm-
entwicklung

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Können wir (von Menschen erschaffener) Software für AKWs, Flugzeuge, Autos, usw. vertrauen?
 - Aktive Forschungsrichtungen innerhalb der Informatik
 - Verbesserung der Testmethoden — keine Garantie für Korrektheit
 - Maschinelle Beweise (d.h. für alle Fälle gültig) der Korrektheit
 - Kein Schutz gegen Fehler in der Spezifikation gegen die geprüft wird!
 - Auch das Beweissystem kann Fehler besitzen.
- Aber wir *reduzieren die Fehlerwahrscheinlichkeit!*
- Heute wird auch über die *probabilistische Korrektheit* nachgedacht und geforscht: Insbesondere die KI kann keine harten Garantien geben

Programm-
entwicklung

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



Zusammenfassung

Programm-
entwicklung

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung



- Fehlerfreie Programmentwicklung gibt es nicht.
- Wir unterscheiden zwischen syntaktischen Fehlern, Laufzeitfehlern und logischen Fehlern.
- Fehlersuche: **Debuggen**
- **Checkliste Ursachenforschung**
- Der Debuggingprozess: Eingabe minimieren, Testfall erstellen, Werte beobachten, Hypothese entwickeln
- Fehler verstehen und beseitigen: **Bugfix**.
- Automatische Tests erhöhen die Qualität von Software: testgetriebene Entwicklung und Regressionstests
- **pytest** ist ein Werkzeug zur Automatisierung von Unittests.

Programm-
entwicklung

Debuggen

Tests

Ausblick:
Fehlerfreies
Programmieren?

Zusammen-
fassung