

# Informatik I

## 9. Programmentwicklung: Testen und Debuggen

Bernhard Nebel

Albert-Ludwigs-Universität Freiburg

15. November 2013

# Programmentwicklung & Fehler

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Syntaktische  
Fehler

Laufzeit-  
Fehler

Semantische  
Fehler

Debuggen

Tests

Ausblick:  
Fehlerfreies  
Program-  
mieren?

# Wie kommen Fehler ins Programm?

- Beim Schreiben von Programmen wird nicht immer alles auf Anhieb **richtig** gemacht.
- Tatsächlich ist ja oft nicht einmal klar, was das „Richtige“ ist.
- Selbst für die klaren Fälle: Schreibfehler, zu kurz gedacht, falsche Annahmen
- Man schätzt, dass rund **50%** des Programmieraufwands für die Identifikation und Beseitigung von Fehlern aufgewendet wird.
- Wichtig: Tools für die Fehlersuche und für die Qualitätskontrolle durch automatisches Testen

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Syntaktische  
Fehler  
Laufzeit-  
Fehler  
Semantische  
Fehler

Debuggen

Tests

Ausblick:  
Fehlerfreies  
Program-  
mieren?

# Arten von möglichen Fehlern

**Syntaktische Fehler:** Bemerkt der Python-Interpreter vor der Ausführung und sind meist einfach zu finden und zu reparieren

**Laufzeit-Fehler:** Während der Ausführung passiert nichts (das Programm hängt) oder es gibt eine **Exception**

**Semantischer Fehler:** Alles läuft, aber die Ausgaben und Aktionen des Programms sind anders als erwartet

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Syntaktische  
Fehler

Laufzeit-  
Fehler

Semantische  
Fehler

Debuggen

Tests

Ausblick:  
Fehlerfreies  
Program-  
mieren?

- Der Interpreter gibt Zeile und Punkt an, an dem der Fehler fest gestellt wurde (und zeigt u.U. die Zeile in IDLE direkt an)
- Das tatsächliche Problem kann aber eine oder sogar mehrere Zeilen vorher liegen!
- Typische mögliche Fehler:
  - Python-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!
- Bei vielen der Probleme helfen Editoren mit Python-Syntaxunterstützung
- Im schlechtesten Fall: Sukzessives Löschen und Probieren

# Laufzeitfehler: Das Programm hängt

- Es wartet auf eine Konsolen-Eingabe (→ Eingabetaste drücken).
- Es wartet auf Daten aus anderer Quelle.
- Es befindet sich in einer Endlosschleife oder Endlosrekursion (d.h. kommt nie zum Basisfall)
  - **Beispiel:** in einer `while`-Schleife wird die Schleifenvariable nicht geändert, z.B. in `tokenizer` das `token`, `linr =` schreiben.

→ Abbrechen mit Ctrl-C!

- Dann Fehler einkreisen und identifizieren (siehe **Debugging**)

# Laufzeitfehler: Exceptions

- 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.
      - **Beispiel**: in `next_token` die `if not ln`-Klausel weglassen.
    - `KeyError`: Ist ähnlich wie `IndexError`, aber für *Dictionaries* (lernen wir noch).
    - `AttributeError`: Ein nicht existentes Attribut wurde versucht anzusprechen (lernen wir noch).
  - Es gibt einen Stack-Backtrace und eine genaue Angabe der Stelle.
- Nachdenken oder Fehler durch Ausgabe von Variablenwerten versuchen zu verstehen
- Dann Fehler einkreisen und identifizieren (siehe **Debugging**).

# Semantische Fehler: Unerfüllte Erwartungen

- Ein semantischer Fehler liegt vor, wenn das Verhalten/die Ausgabe des Programms von der **Erwartung** abweicht, die der Programmier hat.
    - **Beispiel:** Punkt- und Strichrechnung vertauschen
  - Tatsächlich kann man hier eigentlich erst von einem Fehler sprechen, wenn man das erwartete Verhalten **formal spezifiziert** hatte, aber auch informelle Vorgaben können natürlich verletzt werden.
  - Auf jeden Fall kann man das erwartete Verhalten (partiell) durch Beispiele einfach beschreiben.
- Durch Nachdenken versuchen, den relevanten Programmteil zu identifizieren, dann einkreisen (siehe **Debugging**).

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Syntaktische  
Fehler

Laufzeit-  
Fehler

**Semantische  
Fehler**

Debuggen

Tests

Ausblick:  
Fehlerfreies  
Program-  
mieren?



# Debuggen

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

**Debuggen**

Tests

Ausblick:  
Fehlerfreies  
Program-  
mieren?

# 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?)
- Die Verbesserungen heißen **Bugfixes** – und sollten das Problem dann lösen!
- Für das Debugging gibt es verschiedene Methoden:
  - ① Nachdenken (inklusive mentaler Simulation der Programmausführung)
  - ② 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

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Debuggen

Tests

Ausblick:  
Fehlerfreies  
Program-  
mieren?

# Debuggen mit Print-Statements

- Wenn ein System ein abweichendes Verhalten zeigt, versucht man interne Werte zu messen (z.B. bei Hardware mit einem Oszilloskop)
- In Python (und vielen anderen Sprachen/Systemen) kann man einfach `print`-Anweisungen einfügen und das Programm dann laufen lassen.
- Ist die einfachste Möglichkeit, Verhalten eines Programmes zu beobachten, speziell wenn man bereits einen Verdacht hat.
  - **Achtung:** Solche zusätzlichen Ausgaben können natürlich das Verhalten (speziell das Zeitverhalten) signifikant ändern!
- Eine generalisierte Form ist das *Logging*, bei dem man `prints` generell in seinen Code integriert und dann Schalter hat, um das Loggen an- und abzustellen.

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Debuggen

Tests

Ausblick:  
Fehlerfreies  
Program-  
mieren?

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

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

## 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 Programmausführung bzw. setze fort

**Quit:** Beendet alles.

- 1 pdb ist ein Konsolen-orientierter Debugger, der auch Post-Mortem-Analyse anbietet (siehe <http://docs.python.org/3.3/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.
    - Man kann Breakpoints setzen, indem man im Quellcode eine Zeile rechts-klickt (Mac: Ctrl-Klick).
    - Stepping mit den Go/Step usw. Knöpfen.

- 1 Formulieren Sie eine Hypothese, warum der Fehler auftritt und an welcher Stelle des Programms sich dieser Fehler manifestiert!
- 2 Konzentrieren Sie sich auf diese Stelle und **instrumentieren** Sie die Stelle (Breakpoints oder print-Anweisungen)
- 3 Versuchen Sie zu verstehen, wie es zu dem Fehler kommt: Was ist die tiefere Ursache?
- 4 Formulieren Sie einen **Bugfix** erst dann, wenn Sie glauben, das **Problem verstanden** zu haben. Einfache Lösungen sind oft nicht hilfreich.
- 5 Testen Sie nach dem Bugfix, ob das Problem tatsächlich beseitigt wurde.
- 6 Lassen Sie weitere Tests laufen (s.u.).
- 7 Wenn es nicht weiter geht, stehen Sie auf, gehen Sie an die frische Luft und trinken eine Tasse Kaffee!

# Automatische Tests

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Debuggen

**Tests**

Ausblick:  
Fehlerfreies  
Program-  
mieren?

# Testfälle erzeugen

- Um **fehlerhaftes Verhalten** zu provozieren, müssen wir das Programm natürlich testen.
- Man startet das Programm auf Daten (bzw. interagiert) und wartet, bis es **crasht**.
- Am besten systematisch Testfälle sammeln, die man immer wieder für das Programm nutzen kann
- Systematisch Testen:
  - Basisfälle und andere Grenzfälle
  - Decken Sie jeden Zweig in Ihrem Code durch einen Test ab
  - Gibt es Interaktionen zwischen verschiedenen Programmteilen, versuchen Sie auch diese abzudecken
  - **Wichtig:** Tests, die zur Entdeckung eines Fehlers geführt haben, sollten auf jeden Fall für spätere Wiederholungen aufbewahrt werden

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Debuggen

Tests

Ausblick:  
Fehlerfreies  
Program-  
mieren?



- **Regressionstest**: Wiederholung von Tests um sicher zu stellen, dass nach Änderungen der Software keine neuen (oder alten) Fehler eingeschleppt wurden.
- Eine Möglichkeit die **Entwicklung** eines Systems voran zu treiben ist, als erstes Tests zu formulieren, die dann Stück für Stück erfüllt werden
- Die **Qualität** des Systems kann dann mit Hilfe der Anzahl der bestandenen Tests gemessen werden.

- Um zu garantieren, dass die Einzelteile eines System funktionieren, benutzt man sogenannte **Unittests**.
- Diese sind Testfälle für Teile eines Systems (Modul, Funktion, usw.).
- Normalerweise werden diese automatisch ausgeführt.
- In Python gibt es u.a. zwei Werkzeuge/Module:
  - 1 `unittest` - ein komfortables (aber auch aufwändig zu bedienendes) Modul für die Formulierung und Verwaltung von Unit-Tests
  - 2 `doctest` - ein einfaches Modul, das Testfälle aus den docstrings extrahiert und ggfs. automatisch ausführt.

- Fügen Sie Testfälle aus Shellinteraktionen in ihre docstrings ein, z.B. so:

## Testbeispiel

```
import doctest

def tokenizer(line):
    """Takes a string and returns a tuple of tokens.

    >>> tokenizer("")
    ()
    >>> tokenizer("5 ")
    (5,)

    """
    ...
```

# Der `__main__` -Trick

- Nach dem Laden des Programms in IDLE, kann man alle solchen Tests ausführen lassen.

## Python-Interpreter

```
>>> ===== RESTART =====  
>>> doctest.testmode()  
TestResults(failed=0, attempted=30)
```

- Man kann dies automatisieren, indem man am Ende der Datei folgendes hinschreibt (im Erfolgsfall keine Ausgabe):

## Testbeispiel

```
if __name__ == "__main__":  
    doctest.testmod()
```

- Das `__name__`-Attribut ist gleich `"__main__"`, wenn das Modul mit dem Python-Interpreter gestartet wird oder es in IDLE geladen wird.

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Debuggen

Tests

Ausblick:  
Fehlerfreies  
Program-  
mieren?

- Ruft man `doctest.testmod(verbose=True)` auf, bekommt man den Ablauf der Tests angezeigt.
- Will man eine Leerzeile in der Ausgabe der Test-Session haben, so muss man `<BLANKLINE>` eintippen, da eine Leerzeile als Ende des Testfalls interpretiert wird.
- Will oder kann man nicht die gesamte Ausgabe angeben, kann man Auslassungspunkte schreiben: `....`. Dabei muss allerdings ein *Flag* angegeben werden:  
`# doctest: +ELLIPSIS`
- Mehr unter:  
<http://docs.python.org/3.3/library/doctest.html>

# Ausblick: Fehlerfreies Programmieren?

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Debuggen

Tests

**Ausblick:  
Fehlerfreies  
Program-  
mieren?**

# Fehlerfreies Programmieren?

- Können wir (von Menschen erschaffene) Software für AKWs, Flugzeuge, Autos, usw. vertrauen?
  - Testmethoden werden immer besser – decken immer mehr Fälle ab!
  - Manchmal können maschinelle Beweise (d.h. für alle Fälle gültig) die Korrektheit zeigen!
  - Aktive Forschungsrichtung innerhalb der Informatik
  - Natürlich kann aber auch wieder die Spezifikation (gegen die geprüft wird) falsch sein.
  - Auch kann das Beweissystem einen Fehler besitzen.
- Aber wir *reduzieren die Fehlerwahrscheinlichkeit!*
- Heute wird auch über die *probabilistische Korrektheit* nachgedacht und geforscht.

Informatik I

Bernhard  
Nebel

Programm-  
entwicklung  
& Fehler

Debuggen

Tests

Ausblick:  
Fehlerfreies  
Program-  
mieren?