Einführung in die Programmierung
Debugging

Ronja Düffel
WS2017/18

11. Oktober 2017

Debugging

- machen Sie alle schon längst
- *Bug*: Fehler im Programm
- *Debugging*: Suchen und Beheben von Fehlern im Programm
- gefühlt 90% der Zeit beim Programmieren für Fehlersuche
- einige Zahlen:
  - geschätzer wirtschaftlicher Schaden ca 84,4Mrd Euro /jährlich
  - ca 35% des IT-Budgets jährlich für Beseitigung von Programmfehlern
  - ca 70Mrd Euro Produktivitätsverlust durch Computerausfälle

Fehlerarten

- Syntaxfehler: Python versteht nicht, was es tun soll
- Semantischer Fehler: Python versteht was es tun soll, bekommt bei der Ausführung Probleme
- Logische Fehler: Python kann den Code ausführen, liefert aber nicht das gewünschte Ergebnis

Syntaxfehler

- Beispiel in deutscher Sprache: *“Bitte Oma Kuchen.”*
- Verstoß gegen die Syntaxregeln (Grammatik) der Sprache.
- Syntax legt fest
  - welche Zeichen verwendet werden können
  - wie Zeichen zu Ausdrücken zusammengesetzt werden können
  - wie Ausdrücke zu neuen Ausdrücken zusammengesetzt werden können

Rückblick:Syntax der Aussagenlogik

Definition (Syntax der Aussagenlogik)

i) *Jede atomare Aussage ist eine aussagenlogische Formel (aF)*

ii) **0** *ist eine aussagenlogische Formel (aF)*

iii) **1** *ist eine aussagenlogische Formel (aF)*

Rekursive Regeln

iv) *Wenn* φ *eine aF ist, dann ist auch* $\neg\varphi$ *eine aF.*

v) *Wenn* φ *eine aF ist und* ψ *eine aF ist, dann sind*

- $(\varphi \wedge \psi)$,
- $(\varphi \vee \psi)$,
- $(\varphi \rightarrow \psi)$,
- *und* $(\varphi \leftrightarrow \psi)$ *ebenfalls aussagenlogische Formeln.*

Beispiel: Syntaxfehler

- unbekanntes Zeichen

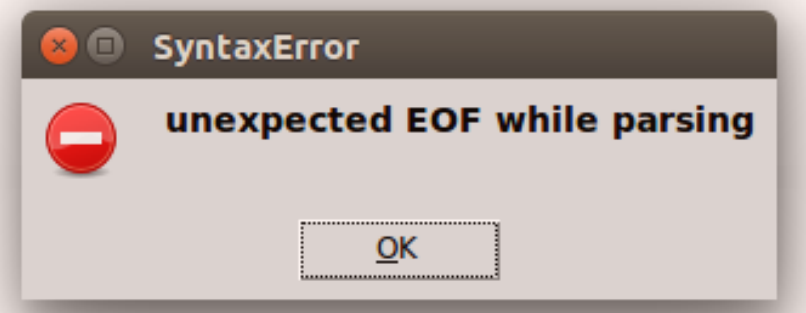
```
>>> 6 ~ 7
SyntaxError: invalid syntax
>>>
```

- fehlender Wert zwischen Operatoren

```
>>> 4+*2
SyntaxError: invalid syntax
>>>
```

- Klammer nicht geschlossen

```
print('Das Ergebnis ist '
```



Beispiel: Syntaxfehler

- fehlender Doppelpunkt

```
>>> for item in liste_1
SyntaxError: invalid syntax
>>>
```

- fehlende schließende Anführungszeichen

```
>>> print('Hi)

SyntaxError: EOL while scanning string literal
>>>
```

- Klammern wo sie nicht hingehören

```
>>> for (item in liste_1):

SyntaxError: invalid syntax
>>>
```

Beispiel: Syntaxfehler

- Fehlende Anführungszeichen

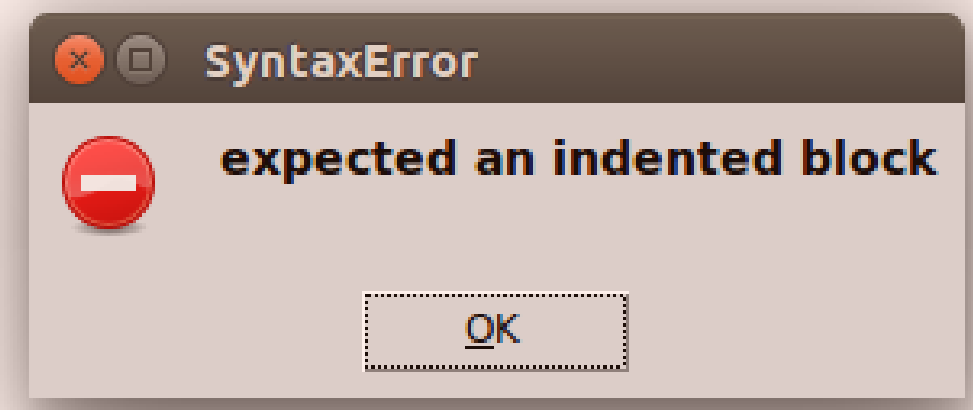
```
>>> print(Hallo!)
SyntaxError: invalid syntax
>>>
```

- nicht erlaubter Variablenname

```
>>> 2Hunde = ["Bello","Lassi"]
SyntaxError: invalid syntax
>>>
```

- falsche Einrückung

```
for i in range(5):
    if i==3:
    continue
    else:
        print(i)
```



Schlüsselwörter

```
>>> class = "Vorkurs"
SyntaxError: invalid syntax
>>>
```

- können nicht als Bezeichner (Variablen-/Funktionsname) verwendet werden
- sind reserviert, da sie bereits ein Objekt/Funktion beschreiben
- Bsp: `not`, `False`, `and`, `while` ...
- Schlüsselwörter anzeigen lassen mit:
  - `import keyword`
  - `print(keyword.kwlist)`

Syntaxfehler

- werden vom Interpreter angezeigt
- werden “gefunden” bevor das Programm startet
- Häufige Syntaxfehler:
  - fehlende Klammern
  - fehlende Anführungszeichen
  - fehlende Kommata
  - fehlender Doppelpunkt

Syntaxfehler vermeiden

- nervig, aber nicht gefährlich, da sie vom Interpreter gefunden werden
- Je mehr man programmiert, desto weniger macht man
- Syntaxhighlighting hilft
- Python-Editor mit automatischer Einrückung

Semantische Fehler

- Beispiel in deutscher Sprache: *“Bitte lauf Oma Kuchen.”*
- Syntaktisch korrekter Python-Ausdruck, aber bei der Ausführung gibt's Probleme

Beispiel: Berechnung der Kreisfläche $A = \pi r^2$

```
1 radius = 5
2 area = pi * radius**2
3 print(area)
```

```
Traceback (most recent call last):
  File "/home/ronja/radius.py", line 4, in <module>
    area = pi * radius**2
NameError: name 'pi' is not defined
>>>
```

`pi` (π) ist eine Konstante, die es nur im `math`-Modul gibt.

TypeError

- Operation/Funktion ist für den Datentyp nicht definiert

Beispiel:

```
1 item = 'notebooks'
2 number = 5
3 store = number + item
```

```
Traceback (most recent call last):
  File "/home/ronja/storage.py", line 3, in <module>
    store = number + item
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

- Häufige Ursache:
  - Benutzereingabe nicht umgewandelt
  - Parameterreihenfolge bei Funktionsaufruf nicht beachtet
  - `return` bei Funktion vergessen, dadurch wird mit `None` weitergearbeitet

Beispiel:Parameterübergabe

```
1 def getNewBalance (now, spent, name):
2     amount = now - spent
3     owner = name
4     return ((owner, amount))
5
6 balance = getNewBalance('Bob', 500, 150)
7 print(balance)
```

```
Traceback (most recent call last):
  File "/media/ronja/Elements/WS1718/Material/Folien/prameter.py", line 6, in <m
odule>
    balance = getNewBalance('Bob', 500, 150)
  File "/media/ronja/Elements/WS1718/Material/Folien/prameter.py", line 2, in ge
tNewBalance
    amount = now - spent
TypeError: unsupported operand type(s) for -: 'str' and 'int'
>>>
```

Parameter benennen

```
1 def getNewBalance (now, spent, name):
2     amount = now - spent
3     owner = name
4     return ((owner, amount))
5
6 balance = getNewBalance(name='Bob', now=500, spent
      =150)
7 print(balance)
```

```
('Bob', 350)
>>>
```

Fehlersuche

- mit `type()` kann man sich den Datentyp einer Variablen ausgeben lassen

```
>>> zahl = '5'
>>> type(zahl)
<class 'str'>
>>>
```

Beispiel type()

```
1 side = input('Seitenlänge in cm: ')
2 area = side**2
3 print('Flächeninhalt:',area)
```

```
1 side = input('Seitenlänge in cm: ')
2 print('debug:',side, type(side))
3 area = side**2
4 print('Flächeninhalt:',area)
```

```
Seitenlänge in cm: 6
debug: 6 <class 'str'>
Traceback (most recent call last):
  File "/home/ronja/Uni/Lernzentrum/Vorkurs/WS1718/Material/Folie
n/typeErrorFind1.py", line 3, in <module>
    area = side**2
TypeError: unsupported operand type(s) for ** or pow(): 'str' and
'int'
>>>
```

NameError

- Verwendung eines Variablennamens oder Funktion, die Python zu dem Zeitpunkt noch nicht kennt
- Häufige Ursache:
  - Tippfehler
  - Variablenname wurde noch nicht definiert
  - Modul wurde nicht importiert
  - Variablenname wird außerhalb des Gültigkeitsbereichs benutzt
  - Funktionsaufruf vor Funktionsdefinition

Beispiel: Gültigkeitsbereich

```
1 def varFunction():
2     var_2 = 5
3     print('varFunction')
4     return
5
6 var_1=5
7 varFunction()
8 var_3 = var_1 + var_2
9 print(var_2,var_3,var_1)
```

```
varFunction
Traceback (most recent call last):
  File "/media/ronja/Elements/WS1718/Material/Folien/nameError.py", line 8, in <
module>
    var_3 = var_1 + var_2
NameError: name 'var_2' is not defined
>>>
```

Fehlersuche/Vermeidung `NameError`

- Suchfunktion des Editors um Tippfehler zu finden
- Einheitliches Schema für Variablennamen/Funktionsnamen
- Autovervollständigung nutzen

ValueError

- Wenn Übergabeparameter außerhalb der von der Funktion erwarteten Parameter liegen.

Beispiel:

```
1 liste = [3,6,1,8]
2 liste.remove(4)
3 print(liste)
```

```
Traceback (most recent call last):
  File "/media/ronja/Elements/WS1718/Material/Folien/valueError.py
", line 2, in <module>
    liste.remove(4)
ValueError: list.remove(x): x not in list
>>>
```

weitere Fehler

IndentationError: Einrückung stimmt nicht (zu viel/wenig Leerzeichen)

IndexError: Versuch auf Index zuzugreifen, der nicht vorhanden ist.

```
>>> liste_1 = ['a','b','c']
>>> liste_1[3]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    liste_1[3]
IndexError: list index out of range
>>>
```

KeyError: Versuch auf nicht vorhanden Eintrag in `dictionary` zuzugreifen

IOError: Versuch auf eine nicht vorhandene Datei zuzufreifen z.B. falscher Name oder Verzeichnis

ZeroDevisionError: Versuch durch 0 zu teilen

- alle Fehlermeldungen kann man in der Python-Dokumentation nachlesen unter:
  https://docs.python.org/3/library/exceptions.html

Logische Fehler

- Beispiel in deutscher Sprache: *"Trink einen Schnaps, das wärmt."*
- Programm läuf ohne Fehlermeldung, aber macht nicht das was es soll.

Beispiel:

```
1 def mittelwert(a,b):
2     '''Berechnet den Mittelwert der übergebenen
          Parameter a und b '''
3     mittelwert=a+b/2
4     return(mittelwert)
```

```
>>> mittelwert(9,2)
10.0
>>>
```

Was tun?

- `print()`-Anweisungen, um herauszufinden was tatsächlich passiert

Beispiel:

```
1  def pow(b, p):
2      y = b ** p
3      return y
4
5  def square(x):
6      a = pow(x, 2)
7      return a
8
9  n = 5
10 result = square(n)
11 print(result)
```

Beispiel:print()

```
1  print(1)
2  def pow(b, p):
3      print(2)
4      y = b ** p
5      print(3)
6      return y
7  print(4)
8  def square(x):
9      print(5)
10     a = pow(x, 2)
11     print(6)
12     return a
13 print(7)
14 n = 5
15 print(8)
16 result = square(n)
17 print(9)
   ('result:',result)
```

Ausgabe

```
1
4
7
8
5
2
3
6
9
result: 25
>>>
```

Was tun bei Fehlern?

- `print()`-**Anweisungen, um herauszufinden was tatsächlich passiert**
  - Wert und Typ von Variablen
  - Nachrichten, wenn bestimmte Stellen/Verzweigungen erreicht werden
- **auskommentieren (zur Not)**
- **Fehler in Schleifen**
  - Endlosschleife (Bedingung nicht verändert?)
  - Index statt Wert benutzt (oder umgekehrt)?
  - nicht alle Indizes benutzt?
    (`range(n)` beginnt bei 0 und hört bei $n-1$ auf)

Was hilft? ("Fehlervermeidung")

- aussagekräftige Namen (Variablen/Funktionen) nach einheitlichem Schema wählen
- unterschiedliche Namen für unterschiedliche Dinge
- Kommentare (#)
- Docstrings (''' '''')
- Mit wenig Code anfangen, nach und nach Code hinzufügen
- Testen

Testen

- Ziel: sicher gehen, dass das Programm tut was wir uns vorgestellt haben
- einzelne Komponenten (Funktionen/Module) testen
- besonderes Augenmerk auf Randbedingungen in Schleifen und Verzweigungen (z.B. leere Listen, sehr große Zahlen, Sonderzeichen in Strings)

Exceptions

- Möglichkeit Fehlermeldungen abzufangen und alternative Ausführung anzugeben
- sehr mächtiges Werkzeug, daher Vorsicht!
- Syntax:

```
try:
    <Anweisung>
except [Art_der_Fehlermeldung]:
    <Alternative_Anweisung>
```

Beispiel: Exceptions

Abfangen von falscher Benutzereingabe

```
1 while True:
2     eing_str = input('Bitte Ganzzahl > 0 eingeben ')
3     try:
4         eing_int = int(eing_str)
5         break
6     except ValueError:
7         print('Das war keine Ganzzahl.')
8 print("Prima! Weiter geht's")
```

```
Bitte Ganzzahl > 0 eingeben neun
Das war keine Ganzzahl.
Bitte Ganzzahl > 0 eingeben 3.5
Das war keine Ganzzahl.
Bitte Ganzzahl > 0 eingeben 4
Prima! Weiter geht's
>>>
```