合肥大學
HEFEI UNIVERSITY

Programming with Python

14. Zwischenspiel: Fehler im Kode mit Exceptions und IDE finden

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

Institute of Applied Optimization (IAO)
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

应用优化研究所
人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2026-01-29

Programming with Python

Dies ist ein Kurs über das Programmieren mit der Programmiersprache Python an der Universität Hefei (合肥大学).

Die Webseite mit dem Lehrmaterial dieses Kurses ist https://thomasweise.github.io/programmingWithPython (siehe auch den QR-Kode unten rechts). Dort können Sie das Kursbuch (in Englisch) und diese Slides finden. Das Repository mit den Beispielprogrammen in Python finden Sie unter https://github.com/thomasWeise/programmingWithPythonCode.

Outline

Einleitung

Einleitung

- In den zukünftigen Slides werden wir eher weniger Screenshots vom PyCharm IDE verwenden.

Einleitung

- In den zukünftigen Slides werden wir eher weniger Screenshots vom PyCharm IDE verwenden.
- Stattdessen werden wir die Programme und ihre Ausgaben als Listings präsentieren.

Einleitung

- In den zukünftigen Slides werden wir eher weniger Screenshots vom PyCharm IDE verwenden.
- Stattdessen werden wir die Programme und ihre Ausgaben als Listings präsentieren.
- Das ist äquivalent mit dem Vorteil, dass wir Kode auch in den Slides selektieren und kopieren können.

Einleitung

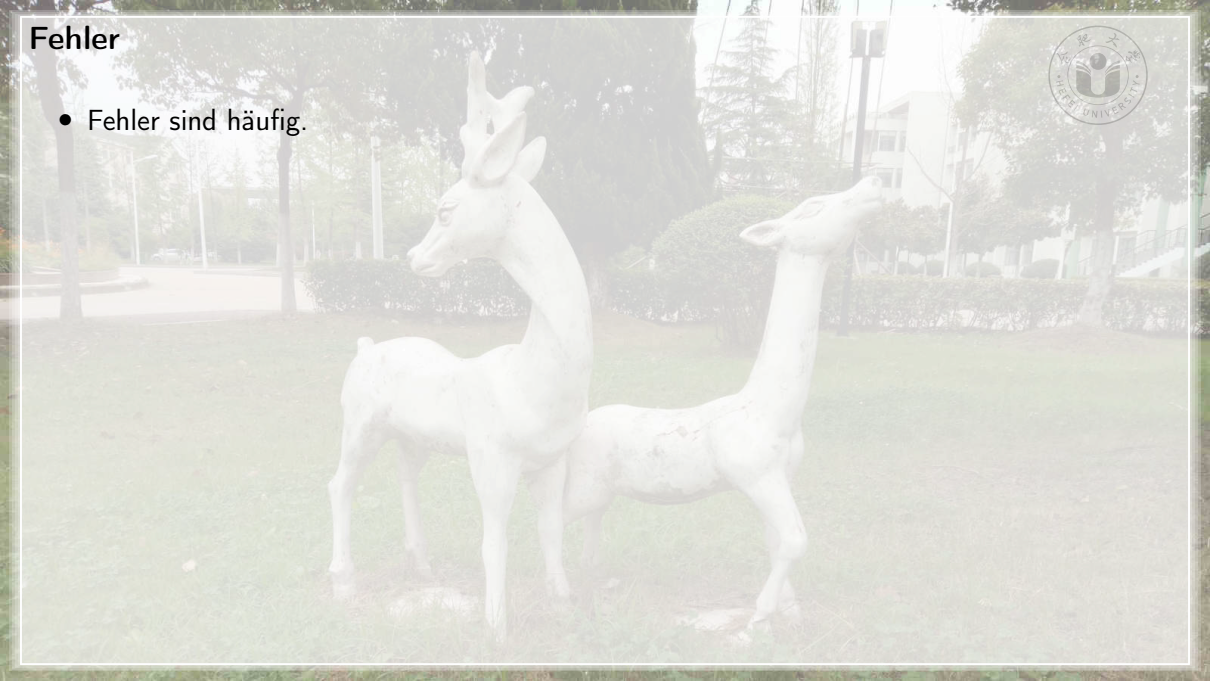
- In den zukünftigen Slides werden wir eher weniger Screenshots vom PyCharm IDE verwenden.
- Stattdessen werden wir die Programme und ihre Ausgaben als Listings präsentieren.
- Das ist äquivalent mit dem Vorteil, dass wir Kode auch in den Slides selektieren und kopieren können.
- Bevor wir uns aber von den Screenshots verabschieden, wollen wir uns noch eine wichtige Funktion anschauen, die viele IDE bereitstellen.

Einleitung

- In den zukünftigen Slides werden wir eher weniger Screenshots vom PyCharm IDE verwenden.
- Stattdessen werden wir die Programme und ihre Ausgaben als Listings präsentieren.
- Das ist äquivalent mit dem Vorteil, dass wir Kode auch in den Slides selektieren und kopieren können.
- Bevor wir uns aber von den Screenshots verabschieden, wollen wir uns noch eine wichtige Funktion anschauen, die viele IDE bereitstellen:
- Sie können uns beim Suchen von Fehlern helfen.

Fehler

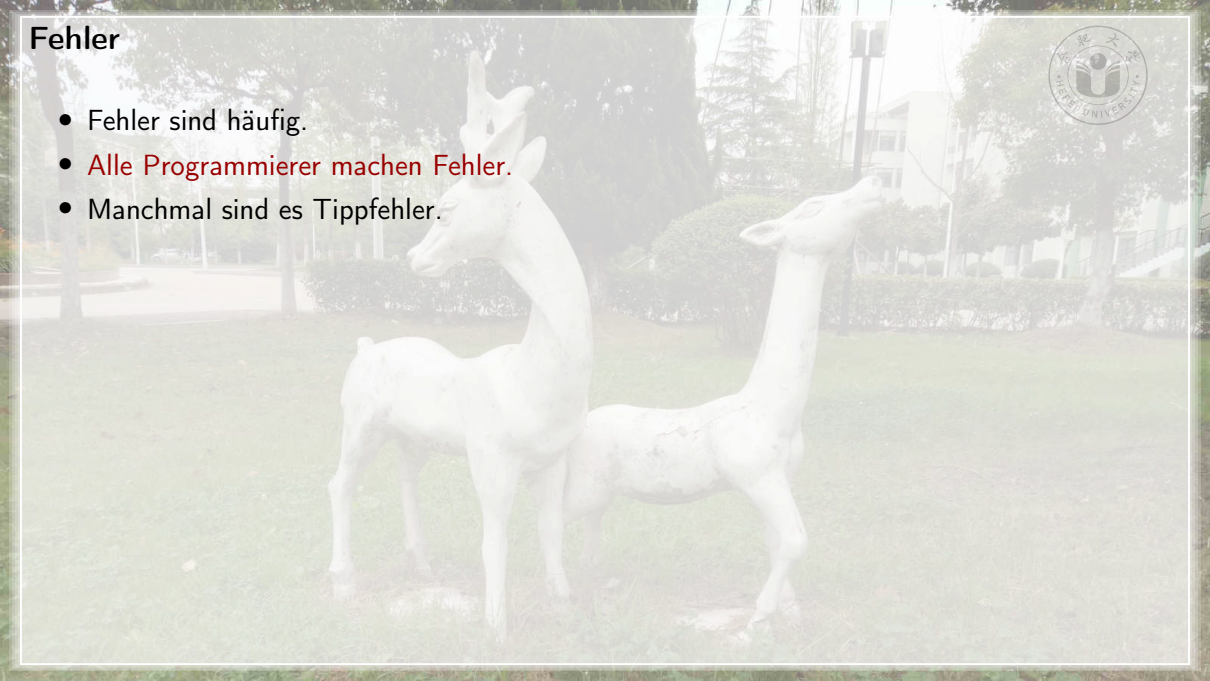
- Fehler sind häufig.



Fehler

- Fehler sind häufig.
- Alle Programmierer machen Fehler.

Fehler



- Fehler sind häufig.
- Alle Programmierer machen Fehler.
- Manchmal sind es Tippfehler.

Fehler

- Fehler sind häufig.
- Alle Programmierer machen Fehler.
- Manchmal sind es Tippfehler. Manchmal kommen wir der Reihenfolge von Parametern einer Funktion durcheinander.

Fehler

- Fehler sind häufig.
- Alle Programmierer machen Fehler.
- Manchmal sind es Tippfehler. Manchmal kommen wir der Reihenfolge von Parametern einer Funktion durcheinander. Manchmal verwechseln wir einen `int` mit einem `float`.

Fehler

- Fehler sind häufig.
- Alle Programmierer machen Fehler.
- Manchmal sind es Tippfehler. Manchmal kommen wir der Reihenfolge von Parametern einer Funktion durcheinander. Manchmal verwechseln wir einen `int` mit einem `float`. Manchmal machen wir einfach logische Fehler.

Fehler

- Fehler sind häufig.
- Alle Programmierer machen Fehler.
- Manchmal sind es Tippfehler. Manchmal kommen wir der Reihenfolge von Parametern einer Funktion durcheinander. Manchmal verwechseln wir einen `int` mit einem `float`. Manchmal machen wir einfach logische Fehler. Manchmal haben wir vielleicht die Funktion, die wir aufrufen, falsch verstanden.

Fehler

- Fehler sind häufig.
- Alle Programmierer machen Fehler.
- Manchmal sind es Tippfehler. Manchmal kommen wir der Reihenfolge von Parametern einer Funktion durcheinander. Manchmal verwechseln wir einen `int` mit einem `float`. Manchmal machen wir einfach logische Fehler. Manchmal haben wir vielleicht die Funktion, die wir aufrufen, falsch verstanden. Manchmal haben wir vielleicht den Algorithmus, den wir gerade versuchen zu implementieren, falsch verstanden.

Fehler

- Fehler sind häufig.
- Alle Programmierer machen Fehler.
- Manchmal sind es Tippfehler. Manchmal kommen wir der Reihenfolge von Parametern einer Funktion durcheinander. Manchmal verwechseln wir einen `int` mit einem `float`. Manchmal machen wir einfach logische Fehler. Manchmal haben wir vielleicht die Funktion, die wir aufrufen, falsch verstanden. Manchmal haben wir vielleicht den Algorithmus, den wir gerade versuchen zu implementieren, falsch verstanden.
- Manche Fehler sind einfach zu finden.

Fehler

- Fehler sind häufig.
- Alle Programmierer machen Fehler.
- Manchmal sind es Tippfehler. Manchmal kommen wir der Reihenfolge von Parametern einer Funktion durcheinander. Manchmal verwechseln wir einen `int` mit einem `float`. Manchmal machen wir einfach logische Fehler. Manchmal haben wir vielleicht die Funktion, die wir aufrufen, falsch verstanden. Manchmal haben wir vielleicht den Algorithmus, den wir gerade versuchen zu implementieren, falsch verstanden.
- Manche Fehler sind einfach zu finden.
- Manchmal führen wir ein Programm aus, as crashed, und die Ausgabe verrät uns, wo der Fehler ist.

Fehler

- Fehler sind häufig.
- Alle Programmierer machen Fehler.
- Manchmal sind es Tippfehler. Manchmal kommen wir der Reihenfolge von Parametern einer Funktion durcheinander. Manchmal verwechseln wir einen `int` mit einem `float`. Manchmal machen wir einfach logische Fehler. Manchmal haben wir vielleicht die Funktion, die wir aufrufen, falsch verstanden. Manchmal haben wir vielleicht den Algorithmus, den wir gerade versuchen zu implementieren, falsch verstanden.
- Manche Fehler sind einfach zu finden.
- Manchmal führen wir ein Programm aus, as crashed, und die Ausgabe verrät uns, wo der Fehler ist.
- Manchmal müssen wir lange debuggen, um die Fehler zu entdecken (das kommt später in Einheit 47).

Fehler

- Fehler sind häufig.
- Alle Programmierer machen Fehler.
- Manchmal sind es Tippfehler. Manchmal kommen wir der Reihenfolge von Parametern einer Funktion durcheinander. Manchmal verwechseln wir einen `int` mit einem `float`. Manchmal machen wir einfach logische Fehler. Manchmal haben wir vielleicht die Funktion, die wir aufrufen, falsch verstanden. Manchmal haben wir vielleicht den Algorithmus, den wir gerade versuchen zu implementieren, falsch verstanden.
- Manche Fehler sind einfach zu finden.
- Manchmal führen wir ein Programm aus, as crashed, und die Ausgabe verrät uns, wo der Fehler ist.
- Manchmal müssen wir lange debuggen, um die Fehler zu entdecken (das kommt später in Einheit 47).
- Bei manchen einfachen Fällen, wie Tippfehlern, kann uns eine gute IDE aber schon helfen.

Fehler mit Hilfe der Ausgabe des Programms finden

Beispiel

- Das Programm `assignment_wrong.py`, eine fehlerhafte Variante des Programmes `assignment.py` aus der vorigen Einheit.

```python
# We define a variable named "int_var" and assign the int value 1 to it.
int_var = 1

# We can use the variable int_var in computations like any other value.
print(2 + int_var)  # This should print 2 + int_var = 2 + 1 = 3.

# We can also use the variable in f-strings.
print(f"int_var has value {int_var}.")  # prints 'int_var has value 1.'

# We can also change the value of the variable.
int_var = (3 * int_var) + 1  # int_var = (3 * 1) + 1 = 4
print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'

float_var = 3.5  # Ofcourse we can also use floating point numbers.
print(f"float_var has value {float_var}.")  # 'float_var has value 3.5.'

new_var = float_var * int_var  # new_var = 3.5 * 4 = 14.0 <- a float!
print(f"{new_var = }.")
```

Beispiel

- Das Programm `assignment_wrong.py`, eine fehlerhafte Variante des Programmes `assignment.py` aus der vorigen Einheit.
- Können Sie den Fehler finden?

```python
# We define a variable named "int_var" and assign the int value 1 to it.
int_var = 1

# We can use the variable int_var in computations like any other value.
print(2 + int_var)  # This should print 2 + int_var = 2 + 1 = 3.

# We can also use the variable in f-strings.
print(f"int_var has value {int_var}.")  # prints 'int_var has value 1.'

# We can also change the value of the variable.
int_var = (3 * int_var) + 1  # int_var = (3 * 1) + 1 = 4
print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'

float_var = 3.5  # Ofcourse we can also use floating point numbers.
print(f"float_var has value {float_var}.")  # 'float_var has value 3.5.'

new_var = float_var * int_var  # new_var = 3.5 * 4 = 14.0 <- a float!
print(f"{new_var = }.")
```

Beispiel

- Können Sie den Fehler finden?
- Genau. In Zeile 12 hat der Programmierer aus Versehen `intvar` statt `int_var` geschrieben.

```
1  # We define a variable named "int_var" and assign the int value 1 to it.
2  int_var = 1
3
4  # We can use the variable int_var in computations like any other value.
5  print(2 + int_var)  # This should print 2 + int_var = 2 + 1 = 3.
6
7  # We can also use the variable in f-strings.
8  print(f"int_var has value {int_var}.")  # prints 'int_var has value 1.'
9
10 # We can also change the value of the variable.
11 int_var = (3 * int_var) + 1  # int_var = (3 * 1) + 1 = 4
12 print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
13
14 float_var = 3.5  # Ofcourse we can also use floating point numbers.
15 print(f"float_var has value {float_var}.")  # 'float_var has value 3.5.'
16
17 new_var = float_var * int_var  # new_var = 3.5 * 4 = 14.0 <- a float!
18 print(f"{new_var = }.")
```

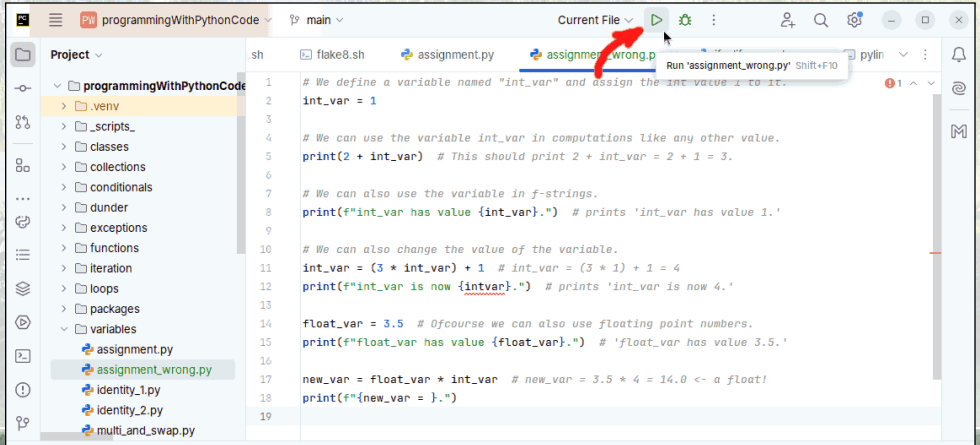
Fehler in der Programmausgabe

- Wenn wir das Programm ausführen, so bekommen wir einen Fehler angezeigt.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                           ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

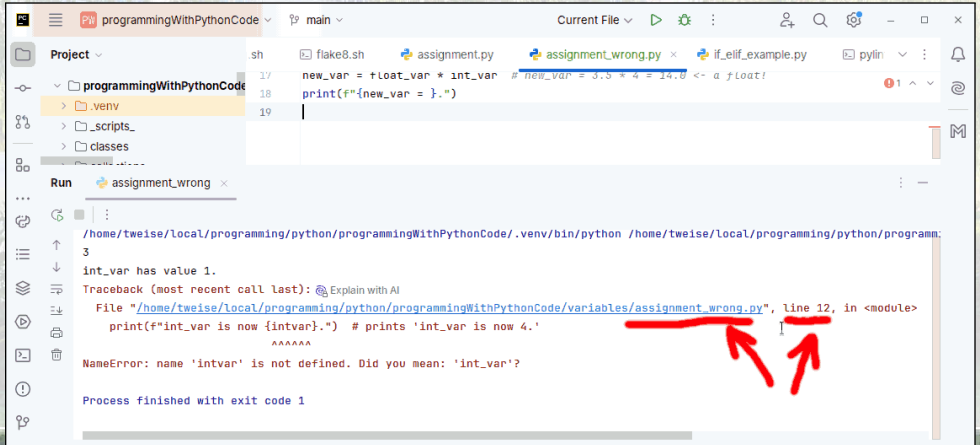
Fehler in der Programmausgabe

- Wenn wir das Programm ausführen, so bekommen wir einen Fehler angezeigt.
- Wir können das Programm auch in dem PyCharm IDE ausführen, in dem wir entweder auf ▷ klicken oder in dem wir ⇧ + F10 drücken.



Fehler in der Programmausgabe

- Wir können das Programm auch in dem PyCharm IDE ausführen, in dem wir entweder auf ▷ klicken oder in dem wir ⇧ + F10 drücken.
- Die Ausgabe stimmt genau mit dem Listing on vorhin überein: Etwas ist schief gegangen!



Fehler in der Programmausgabe

- Die Ausgabe stimmt genau mit dem Listing on vorhin überein: Etwas ist schief gegangen!
- Wenn ein Programm carashed, dann ist das Erste, was wir machen, uns alle Ausgaben sehr genau anzuschauen.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                            ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Die Ausgabe stimmt genau mit dem Listing on vorhin überein: Etwas ist schief gegangen!
- Wenn ein Programm carashed, dann ist das Erste, was wir machen, uns alle Ausgaben sehr genau anzuschauen.
- In unserem Beispiel hier sagt uns der Text sogar ziemlich genau, was schief gegangen ist *und* schlägt sogar vor, wie wir das korrigieren können.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                           ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Die Ausgabe stimmt genau mit dem Listing on vorhin überein: Etwas ist schief gegangen!
- Wenn ein Programm carashed, dann ist das Erste, was wir machen, uns alle Ausgaben sehr genau anzuschauen.
- In unserem Beispiel hier sagt uns der Text sogar ziemlich genau, was schief gegangen ist *und* schlägt sogar vor, wie wir das korrigieren können.
- Da steht: „`NameError`: name '`intvar`' is not defined. Did you mean: '`int_var`'?“

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                           ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Wenn ein Programm carashed, dann ist das Erste, was wir machen, uns alle Ausgaben sehr genau anzuschauen.
- In unserem Beispiel hier sagt uns der Text sogar ziemlich genau, was schief gegangen ist *und* schlägt sogar vor, wie wir das korrigieren können.
- Da steht: „*NameError: name 'intvar' is not defined. Did you mean: 'int_var'?*“
- Das ist ziemlich klar.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                            ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- In unserem Beispiel hier sagt uns der Text sogar ziemlich genau, was schief gegangen ist *und* schlägt sogar vor, wie wir das korrigieren können.
- Da steht: „*`NameError`: name '`intvar`' is not defined. Did you mean: '`int_var`'?*“
- Das ist ziemlich klar.
- Wir haben auf eine Variable (einen Namen) `intvar` zugegriffen, die/den wir nicht definiert oder zugewiesen haben.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                            ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Da steht: „*NameError: name 'intvar' is not defined. Did you mean: 'int_var'?*“
- Das ist ziemlich klar.
- Wir haben auf eine Variable (einen Namen) `intvar` zugegriffen, die/den wir nicht definiert oder zugewiesen haben.
- Diese Variable existiert nicht.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                            ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Das ist ziemlich klar.
- Wir haben auf eine Variable (einen Namen) `intvar` zugegriffen, die/den wir nicht definiert oder zugewiesen haben.
- Diese Variable existiert nicht.
- Der Python-Interpreter hat dann nachgeschlagen, ob es nicht eine Variable mit einem ähnlichen Namen gibt.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                            ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Wir haben auf eine Variable (einen Namen) `intvar` zugegriffen, die/den wir nicht definiert oder zugewiesen haben.
- Diese Variable existiert nicht.
- Der Python-Interpreter hat dann nachgeschlagen, ob es nicht eine Variable mit einem ähnlichen Namen gibt.
- Er hat gefunden, dass es eine Variable namens `int_var` gibt.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                            ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Diese Variable existiert nicht.
- Der Python-Interpreter hat dann nachgeschlagen, ob es nicht eine Variable mit einem ähnlichen Namen gibt.
- Er hat gefunden, dass es eine Variable namens `int_var` gibt.
- Mehrnoch, er teilt uns sogar die genaue Datei und Zeilennummer mit, wo der Fehler passiert ist: in Zeile 12 von Datei `assignment_wrong.py`!

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                           ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Der Python-Interpreter hat dann nachgeschlagen, ob es nicht eine Variable mit einem ähnlichen Namen gibt.
- Er hat gefunden, dass es eine Variable namens `int_var` gibt.
- Mehrnoch, er teilt uns sogar die genaue Datei und Zeilennummer mit, wo der Fehler passiert ist: in Zeile 12 von Datei `assignment_wrong.py`!
- Mit dieser Information haben wir gute Chancen, den Fehler zu finden und zu korrigieren.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                            ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Er hat gefunden, dass es eine Variable namens `int_var` gibt.
- Mehrnoch, er teilt uns sogar die genaue Datei und Zeilennummer mit, wo der Fehler passiert ist: in Zeile 12 von Datei `assignment_wrong.py`!
- Mit dieser Information haben wir gute Chancen, den Fehler zu finden und zu korrigieren.
- Dieser so genannte `Exception` Stack Trace, der auf dem *Standard Error Stream (`stderr`)* ausgegeben wrd, sagt uns also nicht nur, dass es einen Fehler gab.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                            ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Mehrnoch, er teilt uns sogar die genaue Datei und Zeilennummer mit, wo der Fehler passiert ist: in Zeile 12 von Datei `assignment_wrong.py`!
- Mit dieser Information haben wir gute Chancen, den Fehler zu finden und zu korrigieren.
- Dieser so genannte `Exception` Stack Trace, der auf dem *Standard Error Stream (`stderr`)* ausgegeben wrd, sagt uns also nicht nur, dass es einen Fehler gab.
- Er sagt uns auch, was der wahrscheinlichste Grund für den Fehler und wo er wahrscheinlich stattfand.

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                            ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Mit dieser Information haben wir gute Chancen, den Fehler zu finden und zu korrigieren.
- Dieser so genannte `Exception` Stack Trace, der auf dem *Standard Error Stream (`stderr`)* ausgegeben wrd, sagt uns also nicht nur, dass es einen Fehler gab.
- Er sagt uns auch, was der wahrscheinlichste Grund für den Fehler und wo er wahrscheinlich stattfand.
- Wir diskutieren das Thema `Exceptions` später (ab Einheit 31), aber bereits jetzt sollte die Sachse ziemlich klar sein:

```
3
int_var has value 1.
Traceback (most recent call last):
  File "{...}/variables/assignment_wrong.py", line 12, in <module>
    print(f"int_var is now {intvar}.")  # prints 'int_var is now 4.'
                            ^^^^^^
NameError: name 'intvar' is not defined. Did you mean: 'int_var'?
# 'python3 assignment_wrong.py' failed with exit code 1.
```

Fehler in der Programmausgabe

- Dieser so genannte `Exception` Stack Trace, der auf dem *Standard Error Stream (`stderr`)* ausgegeben wrd, sagt uns also nicht nur, dass es einen Fehler gab.
- Er sagt uns auch, was der wahrscheinlichste Grund für den Fehler und wo er wahrscheinlich stattfand.
- Wir diskutieren das Thema `Exceptions` später (ab Einheit 31), aber bereits jetzt sollte die Sachse ziemlich klar sein:

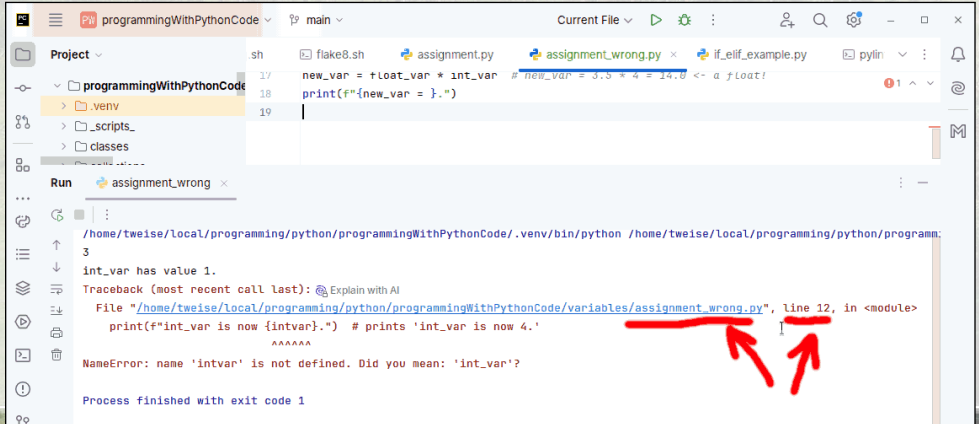
Gute Praxis

Lesen Sie immer alle Fehlermeldungen. Sie geben uns oft wichtige Informationen, wenn wir nach Fehlern suchen. Fehlermeldungen nicht zu lesen ist falsch.

Fehler mit dem IDE suchen

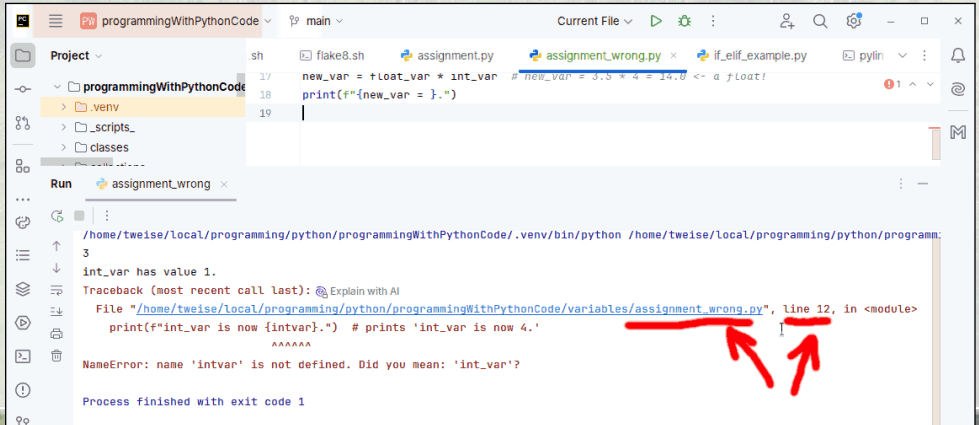
Fehlersuche mit PyCharm

- Wir haben den Fehler gefunden, in dem wir das Programm ausgeführt haben und dann den Output gelesen haben.



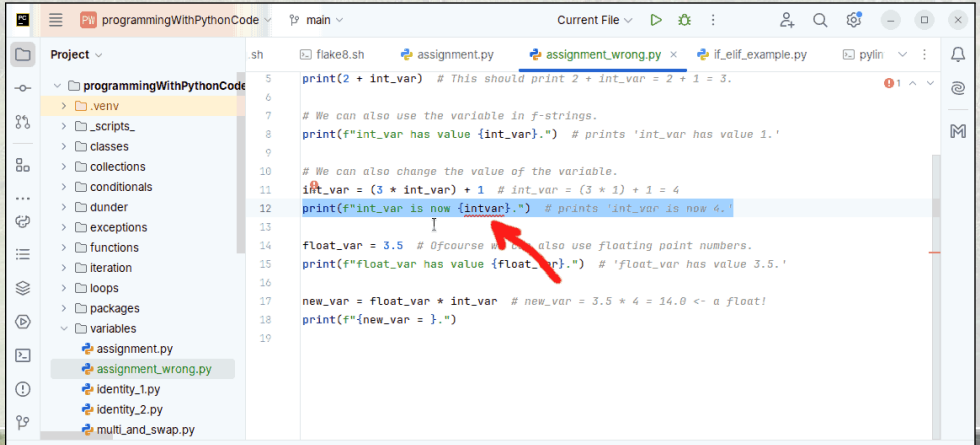
Fehlersuche mit PyCharm

- Wir haben den Fehler gefunden, in dem wir das Programm ausgeführt haben und dann den Output gelesen haben.
- In der Konsole unten in PyCharm, in der die Ausgabe des Programms steht, können wir sogar auf die verlinkte Zeile in der Datei klicken.



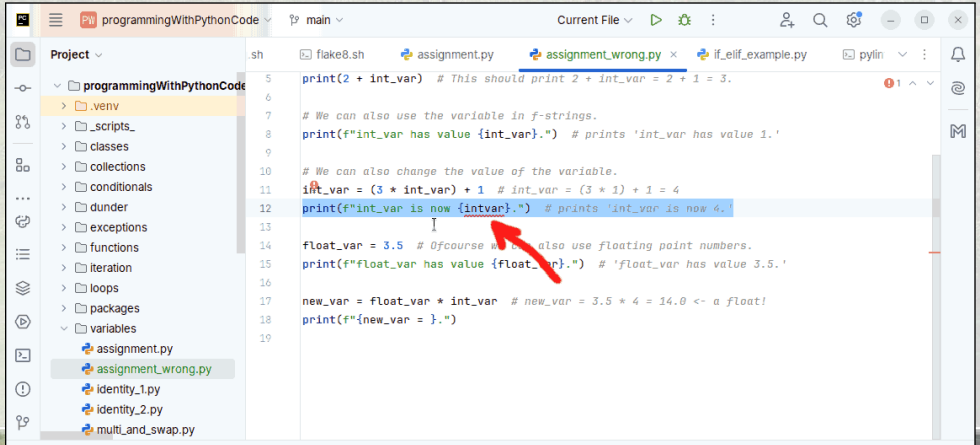
Fehlersuche mit PyCharm

- In der Konsole unten in PyCharm, in der die Ausgabe des Programms steht, können wir sogar auf die verlinkte Zeile in der Datei klicken.
- Das bringt uns direkt zu der Zeile mit dem Fehler.



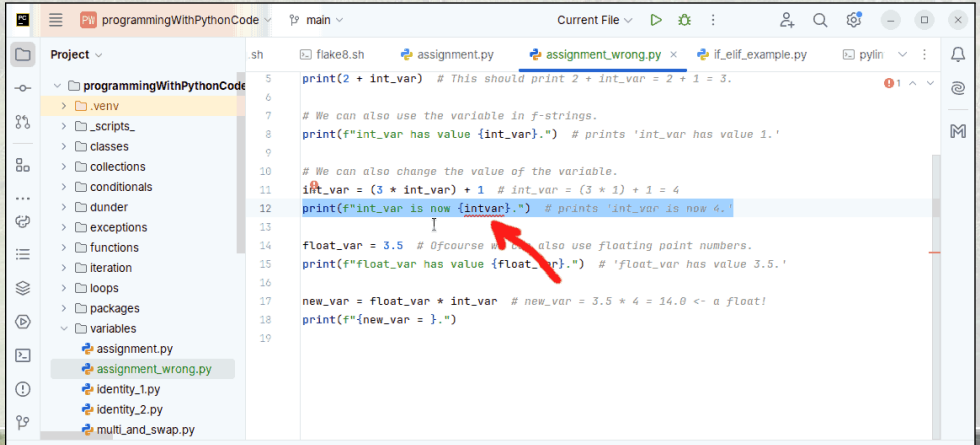
Fehlersuche mit PyCharm

- Das bringt uns direkt zu der Zeile mit dem Fehler.
- Die Frage ist: Hätten wir den Fehler auch finden können, ohne das Programm auszuführen?



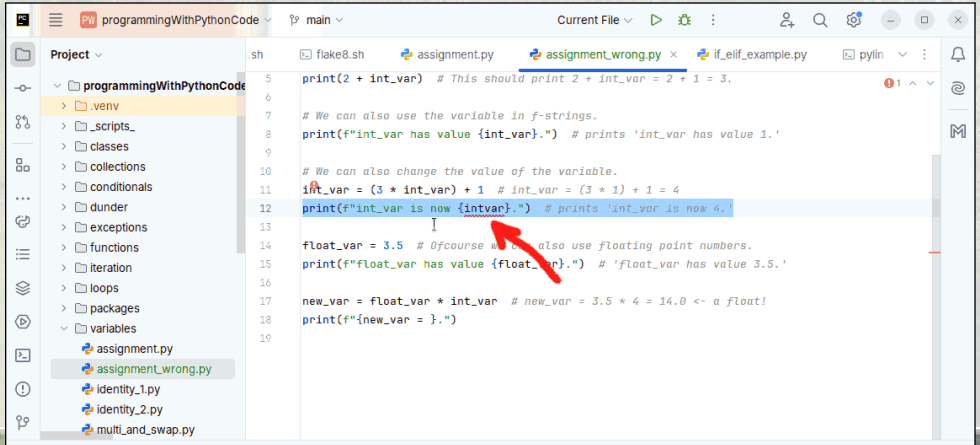
Fehlersuche mit PyCharm

- Die Frage ist: Hätten wir den Fehler auch finden können, ohne das Programm auszuführen?
- Wenn wir uns die Kodezeile genau angucken, dann stellen wir fest, dass das falsch geschriebene eigentlich schon die ganze Zeit mit roter Farbe unterstrichen war!



Fehlersuche mit PyCharm

- Wenn wir uns die Kodezeile genau angucken, dann stellen wir fest, dass das falsch geschriebene eigentlich schon die ganze Zeit mit roter Farbe unterstrichen war!
- Das sollte uns bereits gesagt haben, dass hier irgendetwas nicht stimmt.



Fehlersuche mit PyCharm

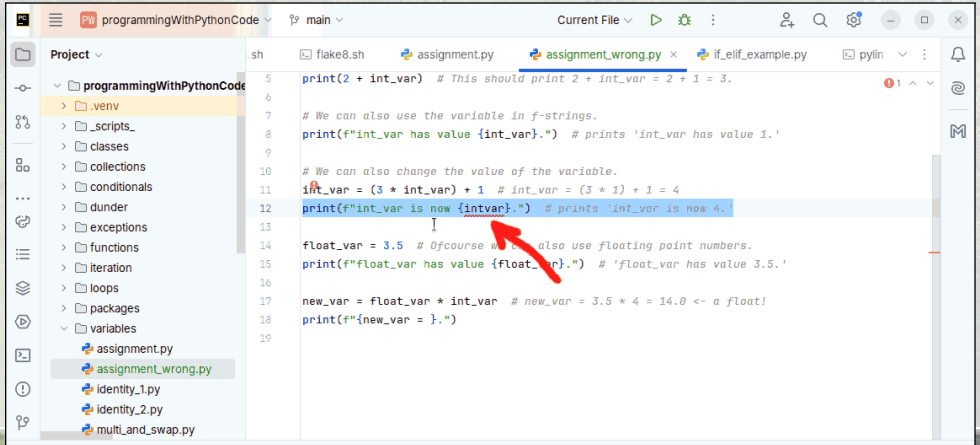
- Das sollte uns bereits gesagt haben, dass hier irgendetwas nicht stimmt.

Gute Praxis

Wenn wir Kode schreiben, sollten wir immer aufpassen, ob das IDE uns über irgendwelche mögliche Fehler benachrichtigt. Bei PyCharm bekommen wir solche Benachrichtigungen oftmals durch rote oder gelbe Unterstreichungen. Wir sollten solche Markierungen immer überprüfen.

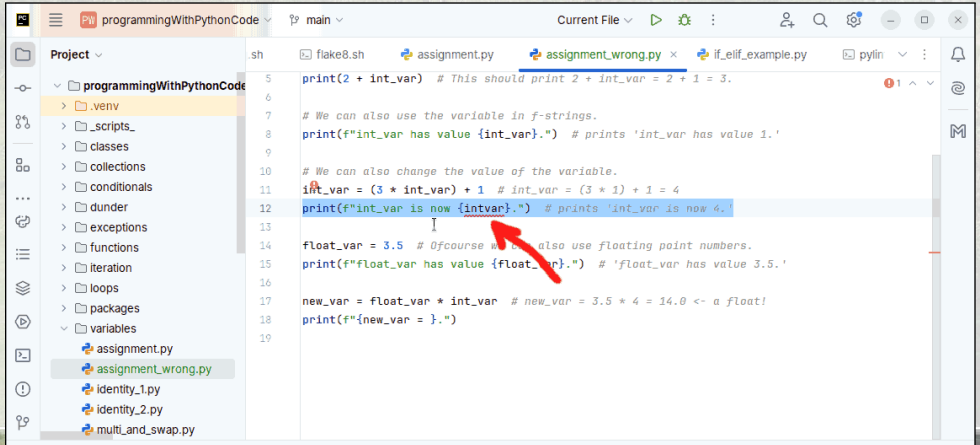
Fehlersuche mit PyCharm

- Das sollte uns bereits gesagt haben, dass hier irgendetwas nicht stimmt.
- Wir kennen jetzt also schon zwei Methoden, Fehler im Kode mit Hilfe des IDE zu finden.



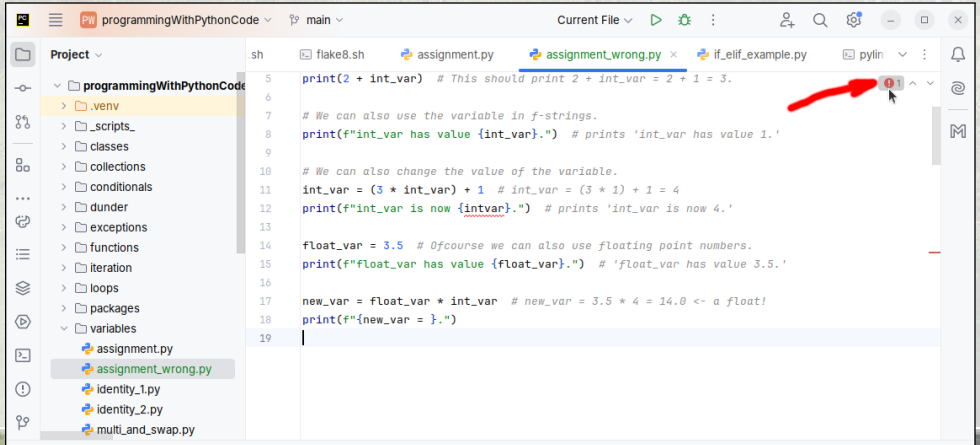
Fehlersuche mit PyCharm

- Wir kennen jetzt also schon zwei Methoden, Fehler im Kode mit Hilfe des IDE zu finden.
- Aber es gibt noch mehr.



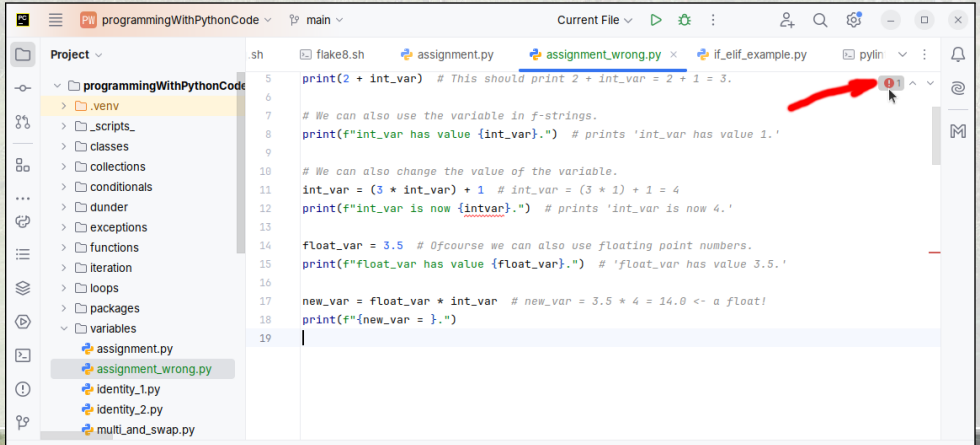
Fehlersuche mit PyCharm

- Aber es gibt noch mehr.
- Die PyCharm IDE zeigt uns auch das kleine rote ❗ Symbol in der oberen rechten Ecke.



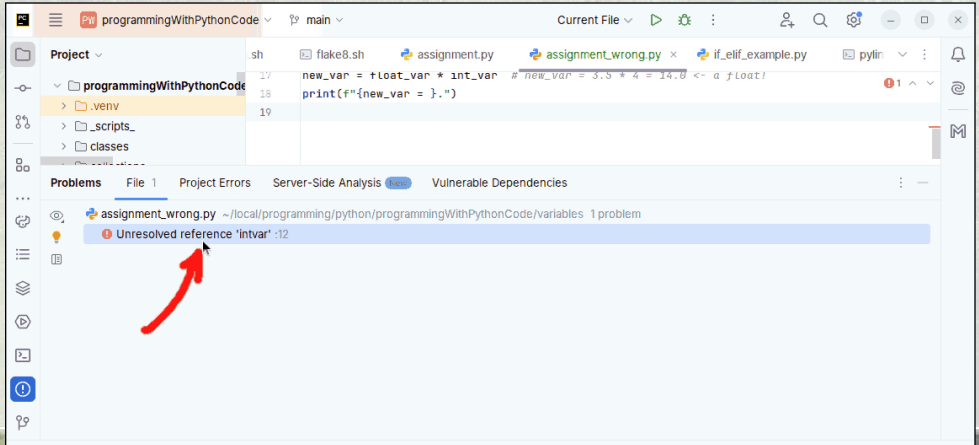
Fehlersuche mit PyCharm

- Die PyCharm IDE zeigt uns auch das kleine rote ❗ Symbol in der oberen rechten Ecke.
- Wenn wir darauf klicken, bekommen wir eine Liste möglicher Fehler und Warnungen angezeigt.



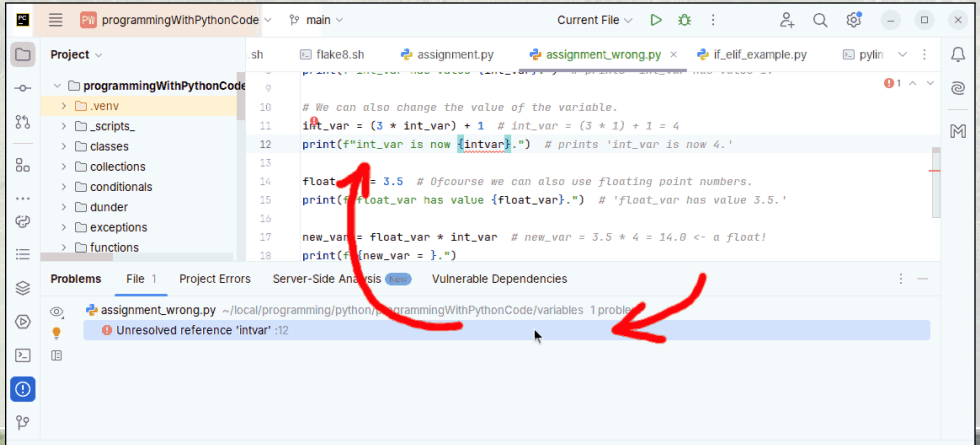
Fehlersuche mit PyCharm

- Wenn wir darauf klicken, bekommen wir eine Liste möglicher Fehler und Warnungen angezeigt.
- Hier sagt uns PyCharm, dass es eine *„Unresolved reference ‘intvar’“* in Zeile 12 gibt.



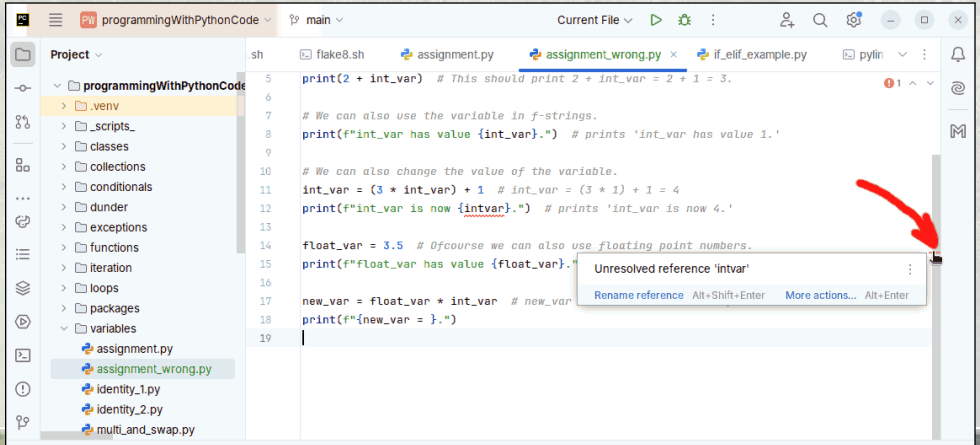
Fehlersuche mit PyCharm

- Hier sagt uns PyCharm, dass es eine *„Unresolved reference 'intvar'"* in Zeile 12 gibt.
- Klicken wir auf diese Nachricht, dann bringt uns das wieder zu der fehlerhaften Zeile.



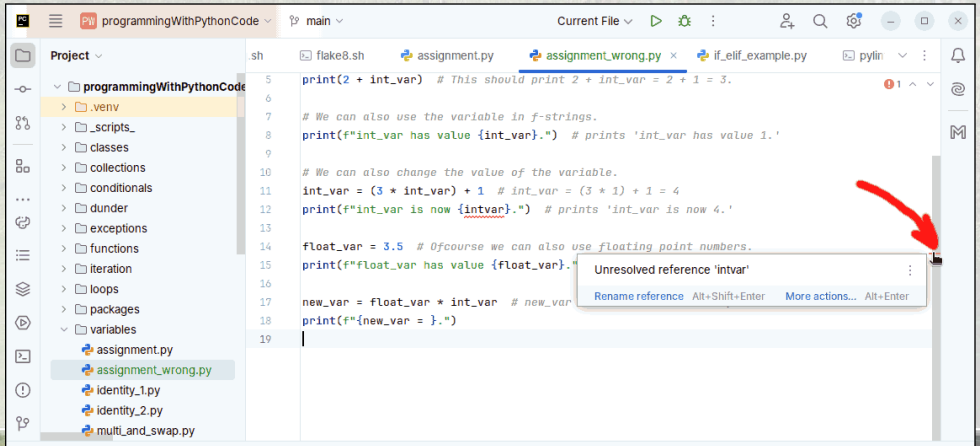
Fehlersuche mit PyCharm

- Klicken wir auf diese Nachricht, dann bringt uns das wieder zu der fehlerhaften Zeile.
- Zusätzlich gibt es auch kleine rote Markierungen am rechten Rand des Editorfensters.



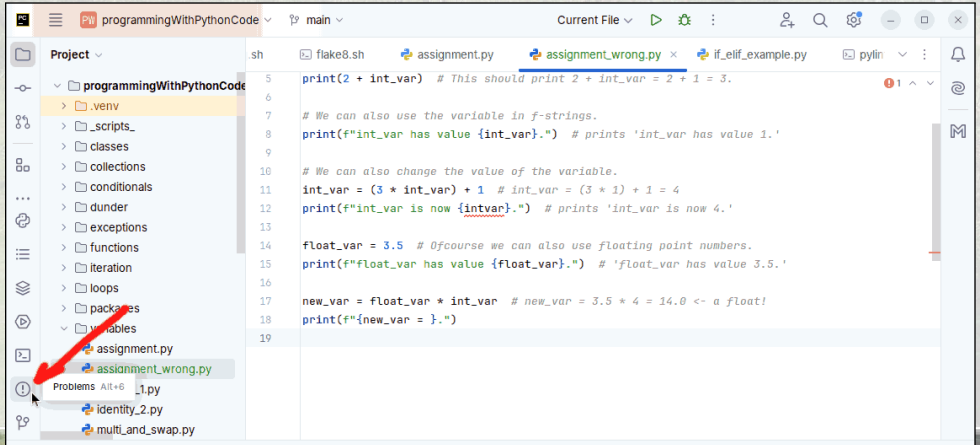
Fehlersuche mit PyCharm

- Zusätzlich gibt es auch kleine rote Markierungen am rechten Rand des Editorfensters.
- Halten wir den Mauskursor über die Markierung, dann öffnet sich eine kleine Ansicht mit der entsprechenden Warnung.



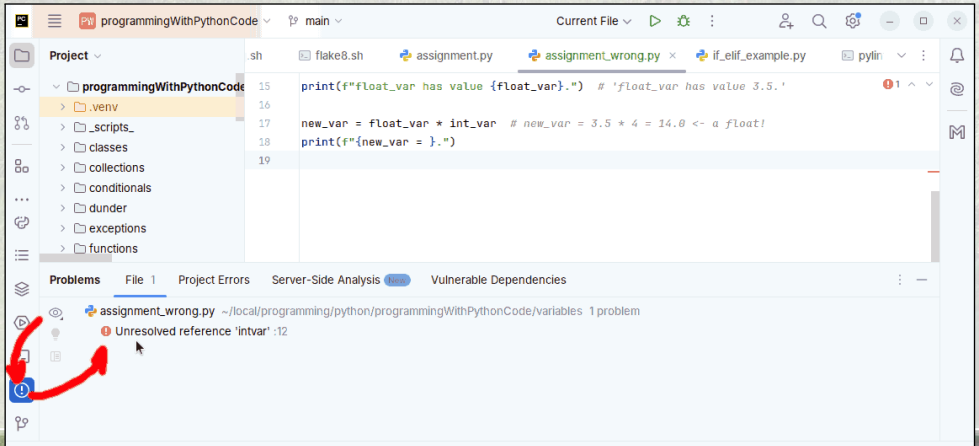
Fehlersuche mit PyCharm

- Wir können auch auf den kleinen ⓘ-Button im Seitenmenü auf der linken Seite klicken oder Alt + 6 drücken.



Fehlersuche mit PyCharm

- Wir können auch auf den kleinen ⓘ-Button im Seitenmenü auf der linken Seite klicken oder Alt + 6 drücken.
- Das bringt uns dann wieder zu der Liste mit möglichen Fehlern.



Fehlersuche mit PyCharm

- Das bringt uns dann wieder zu der Liste mit möglichen Fehlern.

Nützliches Werkzeug

Das IDE und die Fehlermeldungen (`Exception` Stack Traces) sind Ihre wichtigsten Werkzeuge um Fehler zu finden. Lesen Sie die Fehlermeldungen. Gleichgültig ob Sie PyCharm oder irgendein anderes IDE verwenden, wenn dieses IDE Ihren Kode mit Warnungen und Fehlerhinweisen annotiert, dann lesen und prüfen Sie jeden einzelnen Hinweis.

Zusammenfassung

Fehler

- Programmierer machen Fehler.

Fehler

- Programmierer machen Fehler.
- Wir machen Fehler.

Fehler

- Programmierer machen Fehler.
- Wir machen Fehler.
- Sie werden viele Fehler machen.

Fehler

- Programmierer machen Fehler.
- Wir machen Fehler.
- Sie werden viele Fehler machen.
- Jetzt und später.

Fehler

- Programmierer machen Fehler.
- Wir machen Fehler.
- Sie werden viele Fehler machen.
- Jetzt und später.
- Selbst in einfachsten Programmen werden wir Fehler machen.

Fehler

- Programmierer machen Fehler.
- Wir machen Fehler.
- Sie werden viele Fehler machen.
- Jetzt und später.
- Selbst in einfachsten Programmen werden wir Fehler machen.
- Das kann nicht verhindert werden.

Fehler

- Programmierer machen Fehler.
- Wir machen Fehler.
- Sie werden viele Fehler machen.
- Jetzt und später.
- Selbst in einfachsten Programmen werden wir Fehler machen.
- Das kann nicht verhindert werden.
- Die Frage ist also, wie wir damit umgehen.

Fehler

- Programmierer machen Fehler.
- Wir machen Fehler.
- Sie werden viele Fehler machen.
- Jetzt und später.
- Selbst in einfachsten Programmen werden wir Fehler machen.
- Das kann nicht verhindert werden.
- Die Frage ist also, wie wir damit umgehen.
- Wie können wir die Anzahl der Fehler, die wir machen werden, minimieren?

Fehler

- Programmierer machen Fehler.
- Wir machen Fehler.
- Sie werden viele Fehler machen.
- Jetzt und später.
- Selbst in einfachsten Programmen werden wir Fehler machen.
- Das kann nicht verhindert werden.
- Die Frage ist also, wie wir damit umgehen.
- Wie können wir die Anzahl der Fehler, die wir machen werden, minimieren?
- Wie können wir die Anzahl der Fehler, die wir finden und korrigieren, maximieren?

Fehler

- Programmierer machen Fehler.
- Wir machen Fehler.
- Sie werden viele Fehler machen.
- Jetzt und später.
- Selbst in einfachsten Programmen werden wir Fehler machen.
- Das kann nicht verhindert werden.
- Die Frage ist also, wie wir damit umgehen.
- Wie können wir die Anzahl der Fehler, die wir machen werden, minimieren?
- Wie können wir die Anzahl der Fehler, die wir finden und korrigieren, maximieren?
- Die Antwort ist: Mit allen zur Verfügung stehenden Mitteln.

Werkzeuge

- Wir nutzen alle Werkzeuge, die wir finden können, um Fehler zu entdecken und zu korrigieren.

Werkzeuge

- Wir nutzen alle Werkzeuge, die wir finden können, um Fehler zu entdecken und zu korrigieren.
- Wenn wir ein Programm ausführen und es crashed, dann lesen wir Ausgabe genau.

Werkzeuge

- Wir nutzen alle Werkzeuge, die wir finden können, um Fehler zu entdecken und zu korrigieren.
- Wenn wir ein Programm ausführen und es crashed, dann lesen wir Ausgabe genau.
- Wenn unser IDE mögliche Fehler und Warnungen anzeigt, dann schauen wir uns diese genau an.

Werkzeuge

- Wir nutzen alle Werkzeuge, die wir finden können, um Fehler zu entdecken und zu korrigieren.
- Wenn wir ein Programm ausführen und es crashed, dann lesen wir Ausgabe genau.
- Wenn unser IDE mögliche Fehler und Warnungen anzeigt, dann schauen wir uns diese genau an.
- Beides kann unglaublich hilfreich sein.

Werkzeuge

- Wir nutzen alle Werkzeuge, die wir finden können, um Fehler zu entdecken und zu korrigieren.
- Wenn wir ein Programm ausführen und es crashed, dann lesen wir Ausgabe genau.
- Wenn unser IDE mögliche Fehler und Warnungen anzeigt, dann schauen wir uns diese genau an.
- Beides kann unglaublich hilfreich sein.
- Randnotiz: Sie können an der Anzahl der verschiedenen Methoden, wie PyCharm Warnhinweise präsentiert, erkennen, wie wichtig das ist.

Unentdeckte Fehler

- Selbst wenn Ihr Programm wie erwartet funktioniert, dann können trotzdem versteckte Fehler irgendwo im Kode sein.

Unentdeckte Fehler

- Selbst wenn Ihr Programm wie erwartet funktioniert, dann können trotzdem versteckte Fehler irgendwo im Kode sein.
- Manchmal kann man einfach festellen, ob die Ausgabe eines Programms korrekt ist.

Unentdeckte Fehler

- Selbst wenn Ihr Programm wie erwartet funktioniert, dann können trotzdem versteckte Fehler irgendwo im Kode sein.
- Manchmal kann man einfach festellen, ob die Ausgabe eines Programms korrekt ist.
- Manchmal kann man das nicht.

Unentdeckte Fehler

- Selbst wenn Ihr Programm wie erwartet funktioniert, dann können trotzdem versteckte Fehler irgendwo im Kode sein.
- Manchmal kann man einfach festellen, ob die Ausgabe eines Programms korrekt ist.
- Manchmal kann man das nicht.
- Manchmal kann ein Output, der OK aussieht, trotzdem falsch sein.

Unentdeckte Fehler

- Selbst wenn Ihr Programm wie erwartet funktioniert, dann können trotzdem versteckte Fehler irgendwo im Kode sein.
- Manchmal kann man einfach festellen, ob die Ausgabe eines Programms korrekt ist.
- Manchmal kann man das nicht.
- Manchmal kann ein Output, der OK aussieht, trotzdem falsch sein.
- Manchmal können fehlerhafte Befehle in einem Programm sein, die nur bei der aktuellen Ausführung nicht erreicht wurden.

Unentdeckte Fehler

- Selbst wenn Ihr Programm wie erwartet funktioniert, dann können trotzdem versteckte Fehler irgendwo im Kode sein.
- Manchmal kann man einfach festellen, ob die Ausgabe eines Programms korrekt ist.
- Manchmal kann man das nicht.
- Manchmal kann ein Output, der OK aussieht, trotzdem falsch sein.
- Manchmal können fehlerhafte Befehle in einem Programm sein, die nur bei der aktuellen Ausführung nicht erreicht wurden.
- Selbst korrekte Ausgaben garantieren uns nicht, dass unsere Programme korrekt sind.

Unentdeckte Fehler

- Selbst wenn Ihr Programm wie erwartet funktioniert, dann können trotzdem versteckte Fehler irgendwo im Kode sein.
- Manchmal kann man einfach festellen, ob die Ausgabe eines Programms korrekt ist.
- Manchmal kann man das nicht.
- Manchmal kann ein Output, der OK aussieht, trotzdem falsch sein.
- Manchmal können fehlerhafte Befehle in einem Programm sein, die nur bei der aktuellen Ausführung nicht erreicht wurden.
- Selbst korrekte Ausgaben garantieren uns nicht, dass unsere Programme korrekt sind.
- Prüfen Sie daher immer alle Hinweise Ihres IDE.

Unentdeckte Fehler

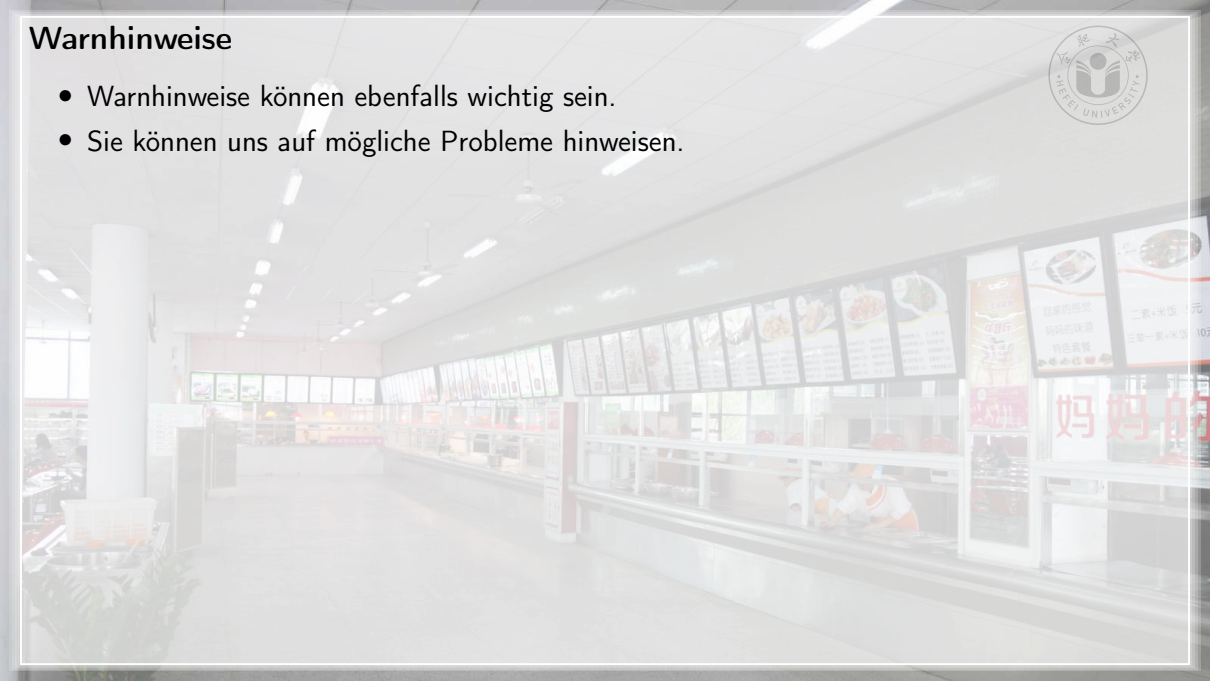
- Selbst wenn Ihr Programm wie erwartet funktioniert, dann können trotzdem versteckte Fehler irgendwo im Kode sein.
- Manchmal kann man einfach festellen, ob die Ausgabe eines Programms korrekt ist.
- Manchmal kann man das nicht.
- Manchmal kann ein Output, der OK aussieht, trotzdem falsch sein.
- Manchmal können fehlerhafte Befehle in einem Programm sein, die nur bei der aktuellen Ausführung nicht erreicht wurden.
- Selbst korrekte Ausgaben garantieren uns nicht, dass unsere Programme korrekt sind.
- Prüfen Sie daher immer alle Hinweise Ihres IDE.
- Stellen Sie sicher, dass Sie immer alle Fehler- und Warnnachrichten vollständig verstanden haben.

Warnhinweise

- Warnhinweise können ebenfalls wichtig sein.

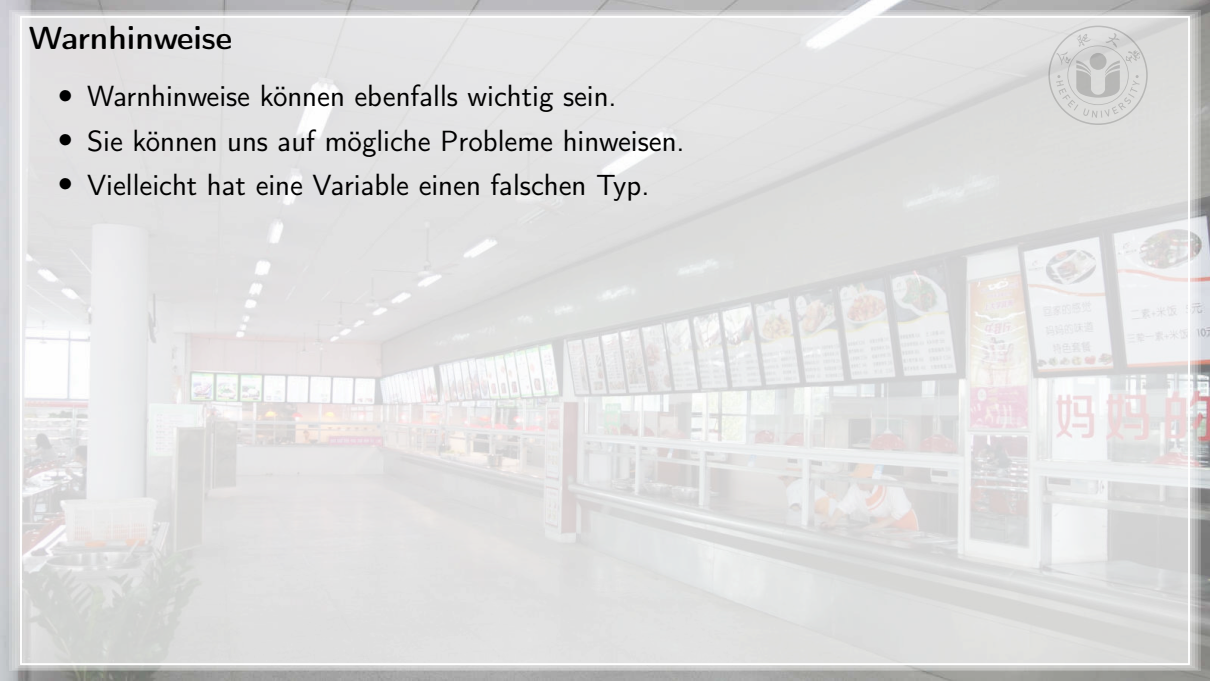
Warnhinweise

- Warnhinweise können ebenfalls wichtig sein.
- Sie können uns auf mögliche Probleme hinweisen.



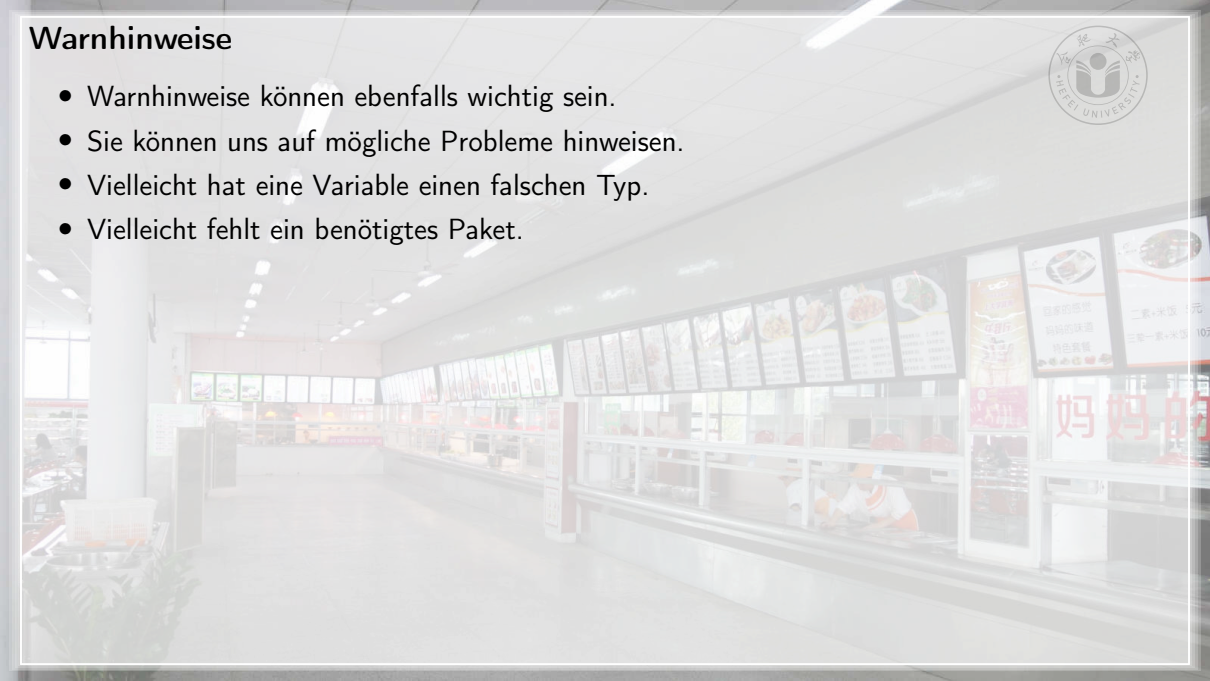
Warnhinweise

- Warnhinweise können ebenfalls wichtig sein.
- Sie können uns auf mögliche Probleme hinweisen.
- Vielleicht hat eine Variable einen falschen Typ.



Warnhinweise

- Warnhinweise können ebenfalls wichtig sein.
- Sie können uns auf mögliche Probleme hinweisen.
- Vielleicht hat eine Variable einen falschen Typ.
- Vielleicht fehlt ein benötigtes Paket.



Warnhinweise

- Warnhinweise können ebenfalls wichtig sein.
- Sie können uns auf mögliche Probleme hinweisen.
- Vielleicht hat eine Variable einen falschen Typ.
- Vielleicht fehlt ein benötigtes Paket.
- Korrigieren Sie alle Fehler und Warnungen wo immer möglich.

Warnhinweise

- Warnhinweise können ebenfalls wichtig sein.
- Sie können uns auf mögliche Probleme hinweisen.
- Vielleicht hat eine Variable einen falschen Typ.
- Vielleicht fehlt ein benötigtes Paket.
- Korrigieren Sie alle Fehler und Warnungen wo immer möglich.
- Selbst wenn Sie denken, dass es False-Positives sind, schauen Sie, ob Sie sie nicht doch korrigieren können.

Warnhinweise

- Warnhinweise können ebenfalls wichtig sein.
- Sie können uns auf mögliche Probleme hinweisen.
- Vielleicht hat eine Variable einen falschen Typ.
- Vielleicht fehlt ein benötigtes Paket.
- Korrigieren Sie alle Fehler und Warnungen wo immer möglich.
- Selbst wenn Sie denken, dass es False-Positives sind, schauen Sie, ob Sie sie nicht doch korrigieren können.
- Denn False-Positives könnten auch von falsch formatiertem Kode herrühren, der für andere Entwickler schwer zu verstehen.

Warnhinweise

- Warnhinweise können ebenfalls wichtig sein.
- Sie können uns auf mögliche Probleme hinweisen.
- Vielleicht hat eine Variable einen falschen Typ.
- Vielleicht fehlt ein benötigtes Paket.
- Korrigieren Sie alle Fehler und Warnungen wo immer möglich.
- Selbst wenn Sie denken, dass es False-Positives sind, schauen Sie, ob Sie sie nicht doch korrigieren können.
- Denn False-Positives könnten auch von falsch formatiertem Kode herrühren, der für andere Entwickler schwer zu verstehen.
- Versuchen Sie immer Warnungs- und fehlerfreien Kode zu produzieren.

Warnhinweise

- Warnhinweise können ebenfalls wichtig sein.
- Sie können uns auf mögliche Probleme hinweisen.
- Vielleicht hat eine Variable einen falschen Typ.
- Vielleicht fehlt ein benötigtes Paket.
- Korrigieren Sie alle Fehler und Warnungen wo immer möglich.
- Selbst wenn Sie denken, dass es False-Positives sind, schauen Sie, ob Sie sie nicht doch korrigieren können.
- Denn False-Positives könnten auch von falsch formatiertem Kode herrühren, der für andere Entwickler schwer zu verstehen.
- Versuchen Sie immer Warnungs- und fehlerfreien Kode zu produzieren.
- Denn, auf der einen Seite, könnten Sie ja falsch liegen, selbst wenn Sie denken, dass die Warnung falsch und der Kode korrekt ist...

Warnhinweise

- Warnhinweise können ebenfalls wichtig sein.
- Sie können uns auf mögliche Probleme hinweisen.
- Vielleicht hat eine Variable einen falschen Typ.
- Vielleicht fehlt ein benötigtes Paket.
- Korrigieren Sie alle Fehler und Warnungen wo immer möglich.
- Selbst wenn Sie denken, dass es False-Positives sind, schauen Sie, ob Sie sie nicht doch korrigieren können.
- Denn False-Positives könnten auch von falsch formatiertem Kode herrühren, der für andere Entwickler schwer zu verstehen.
- Versuchen Sie immer Warnungs- und fehlerfreien Kode zu produzieren.
- Denn, auf der einen Seite, könnten Sie ja falsch liegen, selbst wenn Sie denken, dass die Warnung falsch und der Kode korrekt ist...
- Auf der anderen Seite wird es einfacher, die echten Fehler zu finden, wenn es weniger Warnungen gibt.

Zusammenfassung

- Wir haben nun zwei wichtige Methoden kennengelernt, Fehler in unserem Kode zu finden.

Zusammenfassung

- Wir haben nun zwei wichtige Methoden kennengelernt, Fehler in unserem Kode zu finden.
- Auf der einen Seite gibt uns die Ausgabe bei Programmabstürzen schon ziemlich viele Informationen.

Zusammenfassung

- Wir haben nun zwei wichtige Methoden kennengelernt, Fehler in unserem Kode zu finden.
- Auf der einen Seite gibt uns die Ausgabe bei Programmabstürzen schon ziemlich viele Informationen.
- Auf der anderen Seite können wir viele Hinweise und Warnungen bereits in dem PyCharm IDE sehen.

Zusammenfassung

- Wir haben nun zwei wichtige Methoden kennengelernt, Fehler in unserem Kode zu finden.
- Auf der einen Seite gibt uns die Ausgabe bei Programmabstürzen schon ziemlich viele Informationen.
- Auf der anderen Seite können wir viele Hinweise und Warnungen bereits in dem PyCharm IDE sehen.
- Wichtig ist, dass wir immer allen Warnungen und Fehlermeldungen nachgehen.

Zusammenfassung

- Wir haben nun zwei wichtige Methoden kennengelernt, Fehler in unserem Kode zu finden.
- Auf der einen Seite gibt uns die Ausgabe bei Programmabstürzen schon ziemlich viele Informationen.
- Auf der anderen Seite können wir viele Hinweise und Warnungen bereits in dem PyCharm IDE sehen.
- Wichtig ist, dass wir immer allen Warnungen und Fehlermeldungen nachgehen.
- Wir können nicht verhindern, dass wir Fehler machen.

Zusammenfassung

- Wir haben nun zwei wichtige Methoden kennengelernt, Fehler in unserem Kode zu finden.
- Auf der einen Seite gibt uns die Ausgabe bei Programmabstürzen schon ziemlich viele Informationen.
- Auf der anderen Seite können wir viele Hinweise und Warnungen bereits in dem PyCharm IDE sehen.
- Wichtig ist, dass wir immer allen Warnungen und Fehlermeldungen nachgehen.
- Wir können nicht verhindern, dass wir Fehler machen.
- Aber wir sollten es uns möglichst einfach machen, Fehler zu finden.

谢谢您们！

Thank you!

Vielen Dank!

Glossary (in English) I

Bash is a the shell used under Ubuntu Linux, i.e., the program that „runs“ in the terminal and interprets your commands, allowing you to start and interact with other programs[5,15,32]. Learn more at https://www.gnu.org/software/bash.

debugger A debugger is a tool that lets you execute a program step-by-step while observing the current values of variables. This allows you to find errors in the code more easily[1,19,30]. Learn more about debugging in[29].

IDE An *Integrated Developer Environment* is a program that allows the user do multiple different activities required for software development in one single system. It often offers functionality such as editing source code, debugging, testing, or interaction with a distributed version control system. For Python, we recommend using PyCharm. On Apple systems, Xcode is often used.

iOS is the operating system that powers Apple iPhones[6,24]. Learn more at https://www.apple.com/ios.

Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows[2,8,23,26,28]. We recommend using it for this course, for software development, and for research. Learn more at https://www.linux.org. Its variant Ubuntu is particularly easy to use and install.

macOS or Mac OS is the operating system that powers Apple Mac(intosh) computers[20,24]. Learn more at https://www.apple.com/macos.

Microsoft Windows is a commercial proprietary operating system[4]. It is widely spread, but we recommend using a Linux variant such as Ubuntu for software development and for our course. Learn more at https://www.microsoft.com/windows.

PyCharm is the convenient Python Integrated Development Environment (IDE) that we recommend for this course[27,30,31]. It comes in a free community edition, so it can be downloaded and used at no cost. Learn more at https://www.jetbrains.com/pycharm.

Python The Python programming language[10,13,14,29], i.e., what you will learn about in our book[29]. Learn more at https://python.org.

Glossary (in English) II

stack trace A stack trace gives information the way in which one function invoked another. The term comes from the fact that the data needed to implement function calls is stored in a stack data structure[12]. The data for the most recently invoked function is on top, the data of the function that called is right below, the data of the function that called that one comes next, and so on. Printing a stack trace can be very helpful when trying to find out where an `Exception` occurred.

stderr The *standard error stream* is one of the three pre-defined streams of a console process (together with the standard input stream (`stdin`) and the standard output stream (`stdout`))[11]. It is the text stream to which the process writes information about errors and exceptions. If an uncaught `Exception` is raised in Python and the program terminates, then this information is written to standard error stream (`stderr`). If you run a program in a terminal, then the text that a process writes to its `stderr` appears in the console.

stdin The *standard input stream* is one of the three pre-defined streams of a console process (together with the `stdout` and the `stderr`)[11]. It is the text stream from which the process reads its input text, if any. The Python instruction `input` reads from this stream. If you run a program in a terminal, then the text that you type into the terminal while the process is running appears in this stream.

stdout The *standard output stream* is one of the three pre-defined streams of a console process (together with the `stdin` and the `stderr`)[11]. It is the text stream to which the process writes its normal output. The `print` instruction of Python writes text to this stream. If you run a program in a terminal, then the text that a process writes to its `stdout` appears in the console.

terminal A terminal is a text-based window where you can enter commands and execute them[2,7]. Knowing what a terminal is and how to use it is very essential in any programming- or system administration-related task. If you want to open a terminal under Microsoft Windows, you can Druck auf ⊞+R, dann Schreiben von `cmd`, dann Druck auf ↵. Under Ubuntu Linux, Ctrl+Alt+T opens a terminal, which then runs a Bash shell inside.

Ubuntu is a variant of the open source operating system Linux[7,9]. We recommend that you use this operating system to follow this class, for software development, and for research. Learn more at `https://ubuntu.com`. If you are in China, you can download it from `https://mirrors.ustc.edu.cn/ubuntu-releases`.

Glossary (in English) III

unit test Software development is centered around creating the program code of an application, library, or otherwise useful system. A *unit test* is an *additional* code fragment that is not part of that productive code. It exists to execute (a part of) the productive code in a certain scenario (e.g., with specific parameters), to observe the behavior of that code, and to compare whether this behavior meets the specification[3,16–18,21,25]. If not, the unit test fails. The use of unit tests is at least threefold: First, they help us to detect errors in the code. Second, program code is usually not developed only once and, from then on, used without change indefinitely. Instead, programs are often updated, improved, extended, and maintained over a long time. Unit tests can help us to detect whether such changes in the program code, maybe after years, violate the specification or, maybe, cause another, depending, module of the program to violate its specification. Third, they are part of the documentation or even specification of a program.

Xcode is offers the tools for developing, testing, and distributing applications as well as an IDE for Apple platforms such as macOS and iOS[22].