



Programmieren
lernen mit Python

Konventionen in Python

Kira Grammel, Nina Ihde, Sebastian Serth & Selina Reinhard
Hasso-Plattner-Institut
Universität Potsdam

Motivation

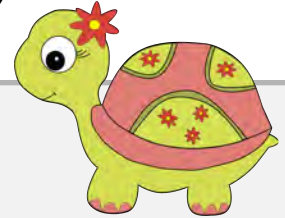
- Programme können mit der Zeit ganz schön umfangreich werden
- In der Regel wird mehr Quellcode gelesen als geschrieben
- Daher gibt es Vereinbarungen zur ...
 - ... Strukturierung des Codes
 - ... Benennung von Funktionen und Variablen
 - ... Erklärung von Codestellen mit Kommentaren
- Code-Formatierung hilft ...
 - ... beim schnellen Erfassen des Codes
 - ... beim Vermeiden von typischen Fehlern
 - ... bei der Zusammenarbeit mit Anderen



Einfachere Lesbarkeit

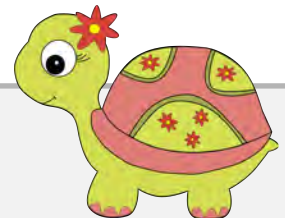
```
1 from daten import position;t=['Schildkröte',"Pinguin",  
2 'Zebra',"Schlange"];l=len(t);print(t[position],l)
```

Schildkröte 4



```
1 from daten import position  
2  
3 tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']  
4 laenge = len(tiere)  
5 print(tiere[position], laenge)
```

Schildkröte 4



Aufbau einer Datei

1. Import am Anfang der Datei

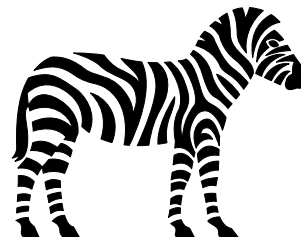
```
from daten import position
```

2. Funktionsdefinition(en)

```
def zufallstier():
    tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']
    laenge = len(tiere)
    print(tiere[position], laenge)
    return tiere[position]
```

3. Aufruf der Funktion(en) sowie Anweisungen außerhalb von Funktionen

```
if zufallstier() == 'Schildkröte':
    print('Hallo Leonie')
```



Leerzeichen im Quellcode

```
1 from daten import position
2
3 def zufallstier():
4     tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']
5     laenge = len(tiere)
6     print(tiere[position], laenge)
7     return tiere[position]
8
9 if zufallstier() == 'Schildkröte':
10     print('Hallo Leonie')
```

Leerzeichen

- Vor und hinter Zuweisungen, Vergleichen und Operatoren (wie `and` und `or`)
- Hinter Kommata bei Aufzählungen (wie in Listen) und bei Funktionen
- Nicht jedoch ...
 - ... um öffnende oder vor schließenden Klammern bei Funktionsaufrufen
 - ... zu Beginn und Ende von Listen
 - ... vor Doppelpunkten

Zeilenumbrüche im Quellcode

```

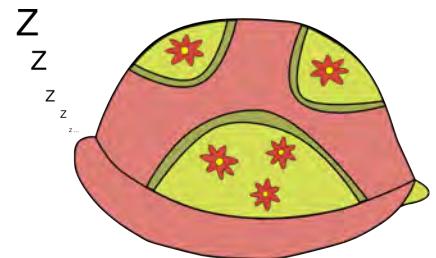
1  from daten import position
2
3  def zufallstier():
4      tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']
5      laenge = len(tiere)
6      print(tiere[position], laenge)
7      return tiere[position]
8
9  if zufallstier() == 'Schildkröte':
10     print('Hallo Leonie')
```

Leerzeilen

- Nach den Import-Anweisungen
- Zwischen Funktionsdefinitionen
- Einzeln zur Strukturierung des Programms

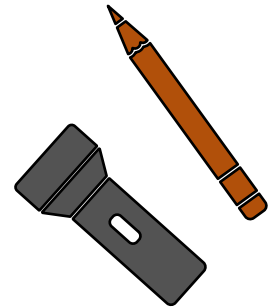
Generell gilt:

- 1 Zeile = 1 Anweisung ohne Semikolon ;



Sinnlose Codezeilen

```
1 from daten import position
2
3 def zufallstier():
4     tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']
5     laenge = len(tiere)
6     laenge = laenge
7     rueckgabewert = tiere[position]
8     return tiere[position]
9     print(tiere[position], laenge)
10
11 if zufallstier() == 'Schildkröte':
12     print('Hallo Leonie')
13     'print(ausgabe)'
```



Unnötige Anweisungen

- Zeile 6 hat keinen Effekt: Zuweisung einer Variable an sich selbst
- Zeile 7 legt eine Variable rueckgabewert an, die nie verwendet wird
- Zeile 9 wird nie ausgeführt, da es nach dem `return` folgt
- Zeile 13 hat ebenfalls keinen Effekt: String ohne Zuweisung

Unabsichtliche Fehlerquellen

```

1  from daten import position
2
3  tiere = ['Leonie', 'Simon', 'Stella']
4
5  def zufallstier():
6      tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']
7      range = len(tiere)
8      print(tiere[position], range)
9      return tiere[position]
10
11 zufallstier()

```

Überschreiben von Namen und Funktionen

- Innerhalb der Funktion `zufallstier` kann (ab Zeile 7) nicht mehr auf die "äußere" Liste `tiere` zugegriffen werden
- Die `range` Methode wird durch eine Zahl ersetzt
- Beides kann aus Versehen passieren; aufwändige Fehlersuche



Verbesserung des Code-Stil

Feedback zum Stil ...

- ... von anderen Programmierer:innen
- ... durch spezielle Programme wie **Linter**
- Linter geben Feedback zu:
 - Code-Formatierungen
 - Sinnlosen Codezeilen
 - Möglichen, unabsichtlichen Fehlerquellen
- Linter können menschliches Feedback nicht ersetzen, aber ergänzen
- Empfehlungen, keine Pflicht



Linten-Feedback auf CodeOcean

The screenshot shows the CodeOcean web interface for an exercise titled "Python20 Aufgabe 3.1.1". The interface includes a code editor on the left with the following Python code:

```
1 from daten import position

def zufallstier():
    tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']
    laenge = len(tiere)
    print
    return

if zufall
print
```

On the right, the "Ergebnisse" (Results) section displays the linter feedback:

Linten-Feedback (ohne Punkte)

Code-Stil	10 von 10
Feedback	Sehr gut. Der Linter hat nichts mehr zu beanstanden.
Meldungen	

Below the feedback, the "Punktzahl: 100%" (Score: 100%) is shown with a green progress bar. At the bottom right, there is a green button labeled "Code zur Bewertung abgeben" (Submit code for evaluation). The footer indicates "Zuletzt gespeichert: 18:58:44" (Last saved: 18:58:44).

Linten-Feedback auf CodeOcean

```
1 from daten import position
2
3 tiere = ['Leonie', 'Simon', 'Stella']
4
5 def zufallstier():
6     tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']
7     range = len(tiere)
8     range = range
9     rueckgabewert = tiere[position]
10    return tiere[position]
11    print(tiere[position], range)
12
13 if zufallstier() == 'Schildkröte':
14     print('Hallo Leonie')
15     'print(ausgabe)'
```

Linten-Feedback auf CodeOcean

```

1 from daten import position
2
3 tiere = ['Leonie', 'Simon', 'Stella']
4
5 def zufallstier():
6     tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']
7     range = len(tiere)
8     range = range

```

Linten-Feedback (ohne Punkte)

Code-Stil

5 von 10

Feedback

Es gibt noch einige Hinweise vom Linter.

Meldungen



- Überschreiben: Der Name 'tiere' wird überschrieben, sodass auf den äußeren Namen (Zeile 3) nicht mehr zugegriffen werden kann
- Überschreiben: Der interne Name 'range' wird überschrieben
- Selbstzuweisung: Die Variable 'range' wird sich selbst zugewiesen
- Unerreichbar: Die Anweisung wird nie ausgeführt werden (Zeile 11)
- Unbenutzt: Die Variable 'rueckgabewert' wird nie verwendet
- Unbenutzt: Der String wird nicht verwendet (Zeile 15)

Linten-Feedback auf CodeOcean

```
1 from daten import position
2
3 tiere = ['Leonie', 'Simon', 'Stella']
4
5 def zufallstier():
6     tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']
7     range = len(tiere)
8     range = range
```

Linten-Feedback (ohne Punkte)


Code-Stil

5 von 10

Feedback

Es gibt noch einige Hinweise vom Linter.

Meldungen

- 
- Überschreiben: Der Name 'tiere' wird überschrieben, sodass auf den äußeren Namen (Zeile 3) nicht mehr zugegriffen werden kann
 - Überschreiben: Der interne Name 'range' wird überschrieben
 - Selbstzuweisung: Die Variable 'range' wird sich selbst zugewiesen
 - Unerreichbar: Die Anweisung wird nie ausgeführt werden (Zeile 11)
 - Unbenutzt: Die Variable 'rueckgabewert' wird nie verwendet
 - Unbenutzt: Der String wird nicht verwendet (Zeile 15)

Linten-Feedback auf CodeOcean

```
1 from daten import position
2
3 tiere = ['Leonie', 'Simon', 'Stella']
4
5 def zufallstier():
6     tiere = ['Schildkröte', 'Pinguin', 'Zebra', 'Schlange']
7     range = len(tiere)
8     range = range
```

Linten-Feedback (ohne Punkte)

Code-Stil

5 von 10

Feedback

Es gibt noch einige Hinweise vom Linter.

Meldungen

- Überschreiben: Der Name 'tiere' wird überschrieben, sodass auf den äußeren Namen (Zeile 3) nicht mehr zugegriffen werden kann
- Überschreiben: Der interne Name 'range' wird überschrieben
- Selbstzuweisung: Die Variable 'range' wird sich selbst zugewiesen
- Unerreichbar: Die Anweisung wird nie ausgeführt werden (Zeile 11)
- Unbenutzt: Die Variable 'rueckgabewert' wird nie verwendet
- Unbenutzt: Der String wird nicht verwendet (Zeile 15)



Linten-Feedback auf CodeOcean

```
6  tiere = [ 'Schildkröte', 'Tigard', 'Zebra', 'Schlange' ]
7  range = len(tiere)
8  range = range
9  rueckgabewert = tiere[position]
10 return tiere[position]
11 print(tiere[position], range)
12
13 if zufallstier() == 'Schildkröte':
14     print('Hallo Leonie')
15     'print(ausgabe)'
```

Linten-Feedback (ohne Punkte)

Code-Stil

5 von 10

Feedback

Es gibt noch einige Hinweise vom Linter.

Meldungen

- Überschreiben: Der Name 'tiere' wird überschrieben, sodass auf den äußeren Namen (Zeile 3) nicht mehr zugegriffen werden kann
- Überschreiben: Der interne Name 'range' wird überschrieben
- Selbstzuweisung: Die Variable 'range' wird sich selbst zugewiesen
- Unerreichbar: Die Anweisung wird nie ausgeführt werden (Zeile 11)
- Unbenutzt: Die Variable 'rueckgabewert' wird nie verwendet
- Unbenutzt: Der String wird nicht verwendet (Zeile 15)



Linten-Feedback auf CodeOcean

```

6   tiere = [ 'Schildkröte', 'Tigard', 'Zebra', 'Schlange' ]
7   range = len(tiere)
8   range = range
9   rueckgabewert = tiere[position]
10  return tiere[position]
11  print(tiere[position], range)
12
13  if zufallstier() == 'Schildkröte':
14      print('Hallo Leonie')
15      'print(ausgabe)'
```

Linten-Feedback (ohne Punkte)

Code-Stil

5 von 10

Feedback

Es gibt noch einige Hinweise vom Linter.

Meldungen

- Überschreiben: Der Name 'tiere' wird überschrieben, sodass auf den äußeren Namen (Zeile 3) nicht mehr zugegriffen werden kann
- Überschreiben: Der interne Name 'range' wird überschrieben
- Selbstzuweisung: Die Variable 'range' wird sich selbst zugewiesen
- Unerreichbar: Die Anweisung wird nie ausgeführt werden (Zeile 11)
- Unbenutzt: Die Variable 'rueckgabewert' wird nie verwendet
- Unbenutzt: Der String wird nicht verwendet (Zeile 15)



Linten-Feedback auf CodeOcean

```

6   tiere = [ 'Schildkröte', 'Tigard', 'Zebra', 'Schlange' ]
7   range = len(tiere)
8   range = range
9   rueckgabewert = tiere[position]
10  return tiere[position]
11  print(tiere[position], range)
12
13  if zufallstier() == 'Schildkröte':
14      print('Hallo Leonie')
15      'print(ausgabe)'
```

Linten-Feedback (ohne Punkte)

Code-Stil

5 von 10

Feedback

Es gibt noch einige Hinweise vom Linter.

Meldungen

- Überschreiben: Der Name 'tiere' wird überschrieben, sodass auf den äußeren Namen (Zeile 3) nicht mehr zugegriffen werden kann
- Überschreiben: Der interne Name 'range' wird überschrieben
- Selbstzuweisung: Die Variable 'range' wird sich selbst zugewiesen
- Unerreichbar: Die Anweisung wird nie ausgeführt werden (Zeile 11)
- Unbenutzt: Die Variable 'rueckgabewert' wird nie verwendet
- Unbenutzt: Der String wird nicht verwendet (Zeile 15)



- Konventionen...
 - ... sorgen für einheitlichen Code
 - ... sind (nur) Empfehlungen und keine Verpflichtungen
 - ... erleichtern die Zusammenarbeit verschiedener Entwickler:innen
- Linter...
 - ... helfen bei der Einhaltung von Konventionen
 - ... geben ein Maß für die Qualität von Quellcode an
 - ... zeigen mögliche Fehlerquellen auf

