



合肥大學
HEFEI UNIVERSITY



Programming with Python

13. Variablen: Wertzuweisung

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

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-Code 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

1. Einleitung
2. Variablen und Kommentare
3. Variablen Zuweisen
4. Kode und Stil
5. Zusammenfassung





Einleitung



Einleitung



- Wir haben bereits verschiedene Datentypen in Python kennengelernt.

Einleitung



- Wir haben bereits verschiedene Datentypen in Python kennengelernt.
- Wir wissen, wie man Ausdrücke schreibt, die z. B. eine mathematische Berechnung durchführen oder Zeichenketten bearbeiten.

Einleitung



- Wir haben bereits verschiedene Datentypen in Python kennengelernt.
- Wir wissen, wie man Ausdrücke schreibt, die z. B. eine mathematische Berechnung durchführen oder Zeichenketten bearbeiten.
- Wir sind aber noch einiges davon entfernt, richtige Programme zu schreiben.

Einleitung



- Wir haben bereits verschiedene Datentypen in Python kennengelernt.
- Wir wissen, wie man Ausdrücke schreibt, die z. B. eine mathematische Berechnung durchführen oder Zeichenketten bearbeiten.
- Wir sind aber noch einiges davon entfernt, richtige Programme zu schreiben.
- Ausdrücke sind oft Berechnungen, die im Großen und Ganzen in einem Schritt ausgeführt werden und mit nur eine Zeile Code definiert werden.

Einleitung



- Wir haben bereits verschiedene Datentypen in Python kennengelernt.
- Wir wissen, wie man Ausdrücke schreibt, die z. B. eine mathematische Berechnung durchführen oder Zeichenketten bearbeiten.
- Wir sind aber noch einiges davon entfernt, richtige Programme zu schreiben.
- Ausdrücke sind oft Berechnungen, die im Großen und Ganzen in einem Schritt ausgeführt werden und mit nur eine Zeile Code definiert werden.
- Für kompliziertere Berechnungen müssten wir in der Lage sein, irgendwie Werte zu speichern.

Einleitung



- Wir haben bereits verschiedene Datentypen in Python kennengelernt.
- Wir wissen, wie man Ausdrücke schreibt, die z. B. eine mathematische Berechnung durchführen oder Zeichenketten bearbeiten.
- Wir sind aber noch einiges davon entfernt, richtige Programme zu schreiben.
- Ausdrücke sind oft Berechnungen, die im Großen und Ganzen in einem Schritt ausgeführt werden und mit nur eine Zeile Code definiert werden.
- Für kompliziertere Berechnungen müssten wir in der Lage sein, irgendwie Werte zu speichern.
- Dafür gibt es **Variable**.



Variablen und Kommentare



Variablen Definieren und Werte Zuweisen



- Wie in der Mathematik ist eine Variable in Python im Grunde nur ein Name, dem ein Wert zugewiesen wurde.

Variablen Definieren und Werte Zuweisen



- Wie in der Mathematik ist eine Variable in Python im Grunde nur ein Name, dem ein Wert zugewiesen wurde.
- Wir können Variablen definieren und ihnen Werte zuweisen, in dem wir schreiben `Name = Wert`.

Variablen Definieren und Werte Zuweisen



- Wie in der Mathematik ist eine Variable in Python im Grunde nur ein Name, dem ein Wert zugewiesen wurde.
- Wir können Variablen definieren und ihnen Werte zuweisen, in dem wir schreiben `Name = Wert`.
- Hier ist `Name` der Name der Variablen und `Wert` ist der Wert, den wir dem Name zuweisen.

Variablen Definieren und Werte Zuweisen



- Wie in der Mathematik ist eine Variable in Python im Grunde nur ein Name, dem ein Wert zugewiesen wurde.
- Wir können Variablen definieren und ihnen Werte zuweisen, in dem wir schreiben `Name = Wert`.
- Hier ist `Name` der Name der Variablen und `Wert` ist der Wert, den wir dem Name zuweisen.
- Wenn wir den gespeicherten Wert benutzen wollen, dann brauchen wir stattdessen nur `Name` zu schreiben.

Variablen Definieren und Werte Zuweisen



- Wie in der Mathematik ist eine Variable in Python im Grunde nur ein Name, dem ein Wert zugewiesen wurde.
- Wir können Variablen definieren und ihnen Werte zuweisen, in dem wir schreiben `Name = Wert`.
- Hier ist `Name` der Name der Variablen und `Wert` ist der Wert, den wir dem Name zuweisen.
- Wenn wir den gespeicherten Wert benutzen wollen, dann brauchen wir stattdessen nur `Name` zu schreiben.
- Wir können `Name` in beliebigen Ausdrücken verwenden und Python benutzt dann stattdessen `Wert`.

Variablen Definieren und Werte Zuweisen



- Wie in der Mathematik ist eine Variable in Python im Grunde nur ein Name, dem ein Wert zugewiesen wurde.
- Wir können Variablen definieren und ihnen Werte zuweisen, in dem wir schreiben `Name = Wert`.
- Hier ist `Name` der Name der Variablen und `Wert` ist der Wert, den wir dem Name zuweisen.
- Wenn wir den gespeicherten Wert benutzen wollen, dann brauchen wir stattdessen nur `Name` zu schreiben.
- Wir können `Name` in beliebigen Ausdrücken verwenden und Python benutzt dann stattdessen `Wert`.
- Das haben Sie sogar schon gesehen, als wir nämlich die Variablen `pi`, `e`, `inf`, und `nan` verwendet haben, die wir aus dem Modul `math` importiert haben.

Variablen Definieren und Werte Zuweisen



- Wie in der Mathematik ist eine Variable in Python im Grunde nur ein Name, dem ein Wert zugewiesen wurde.
- Wir können Variablen definieren und ihnen Werte zuweisen, in dem wir schreiben `Name = Wert`.
- Hier ist `Name` der Name der Variablen und `Wert` ist der Wert, den wir dem Name zuweisen.
- Wenn wir den gespeicherten Wert benutzen wollen, dann brauchen wir stattdessen nur `Name` zu schreiben.
- Wir können `Name` in beliebigen Ausdrücken verwenden und Python benutzt dann stattdessen `Wert`.
- Das haben Sie sogar schon gesehen, als wir nämlich die Variablen `pi`, `e`, `inf`, und `nan` verwendet haben, die wir aus dem Modul `math` importiert haben.
- Sie können auch den Wert ändern, den eine Variable speichert, in dem Sie der Variablen einfach einen neuen Wert zuweise, z. B. `Name = Neuer_Wert`.

Programme

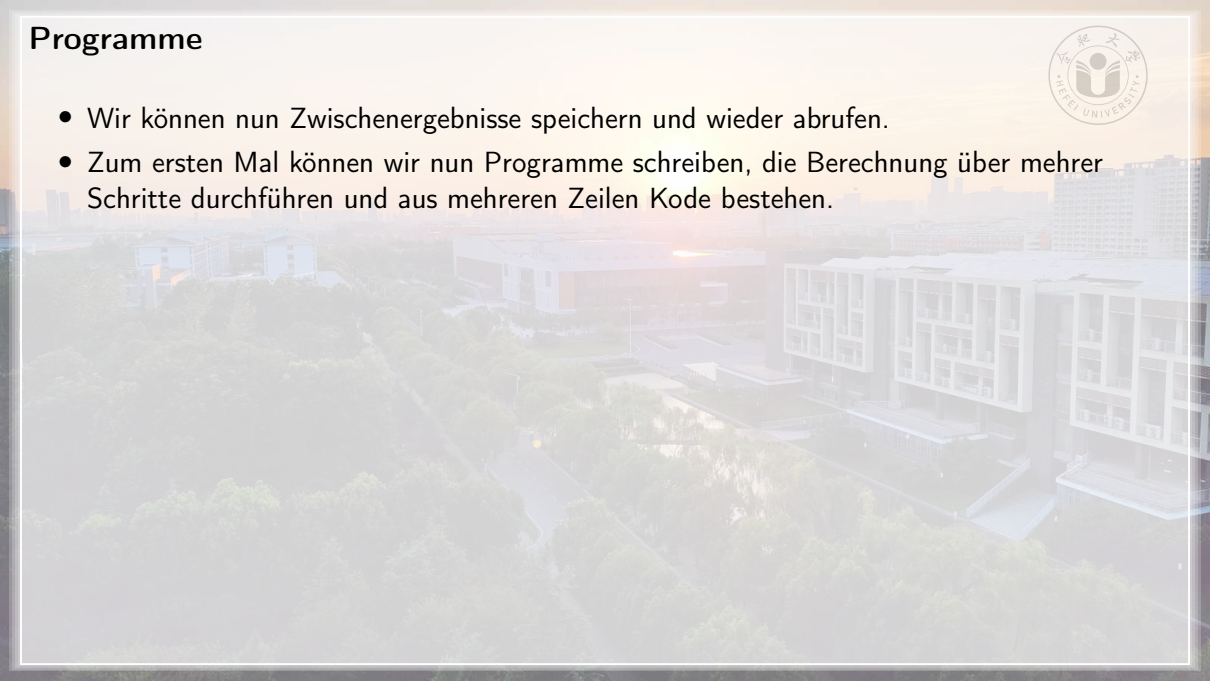
- Wir können nun Zwischenergebnisse speichern und wieder abrufen.



Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrer Schritte durchführen und aus mehreren Zeilen Kode bestehen.



Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrer Schritte durchführen und aus mehreren Zeilen Code bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeichert, z. B. `hallo.py`.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrer Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeichert, z. B. `hallo.py`.
- In genau dem Moment, in dem wir damit beginnen echte Programme zu schreiben, passieren zwei Dinge.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrer Schritte durchführen und aus mehreren Zeilen Code bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeichert, z. B. `hallo.py`.
- In genau dem Moment, in dem wir damit beginnen echte Programme zu schreiben, passieren zwei Dinge:
 1. Unser Code wird viel komplexer.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrer Schritte durchführen und aus mehreren Zeilen Code bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeichert, z. B. `hallo.py`.
- In genau dem Moment, in dem wir damit beginnen echte Programme zu schreiben, passieren zwei Dinge:
 1. Unser Code wird viel komplexer.
 2. Unser Code kann wieder verwendet werden, also mehrmals ausgeführt werden.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrer Schritte durchführen und aus mehreren Zeilen Code bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeichert, z. B. `hallo.py`.
- In genau dem Moment, in dem wir damit beginnen echte Programme zu schreiben, passieren zwei Dinge:
 1. Unser Code wird viel komplexer.
 2. Unser Code kann wieder verwendet werden, also mehrmals ausgeführt werden. Von uns, heute, morgen, oder in zehn Jahren.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrer Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeichert, z. B. `hallo.py`.
- In genau dem Moment, in dem wir damit beginnen echte Programme zu schreiben, passieren zwei Dinge:
 1. Unser Kode wird viel komplexer.
 2. Unser Kode kann wieder verwendet werden, also mehrmals ausgeführt werden. Von uns, heute, morgen, oder in zehn Jahren. Und mit anderen Leuten geteilt werden, die den Kode dann heute, morgen, oder in zehn Jahren ausführen.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrer Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeichert, z. B. `hallo.py`.
- In genau dem Moment, in dem wir damit beginnen echte Programme zu schreiben, passieren zwei Dinge:
 1. Unser Kode wird viel komplexer.
 2. Unser Kode kann wieder verwendet werden, also mehrmals ausgeführt werden. Von uns, heute, morgen, oder in zehn Jahren. Und mit anderen Leuten geteilt werden, die den Kode dann heute, morgen, oder in zehn Jahren ausführen.
- Das ändert die Qualität unseres Kodes extrem stark.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrer Schritte durchführen und aus mehreren Zeilen Code bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeichert, z. B. `hallo.py`.
- In genau dem Moment, in dem wir damit beginnen echte Programme zu schreiben, passieren zwei Dinge:
 1. Unser Code wird viel komplexer.
 2. Unser Code kann wieder verwendet werden, also mehrmals ausgeführt werden. Von uns, heute, morgen, oder in zehn Jahren. Und mit anderen Leuten geteilt werden, die den Code dann heute, morgen, oder in zehn Jahren ausführen.
- Das ändert die Qualität unseres Codes extrem stark.
- Bisher war der Python-Interpreter im Grunde ein Taschenrechner für uns.

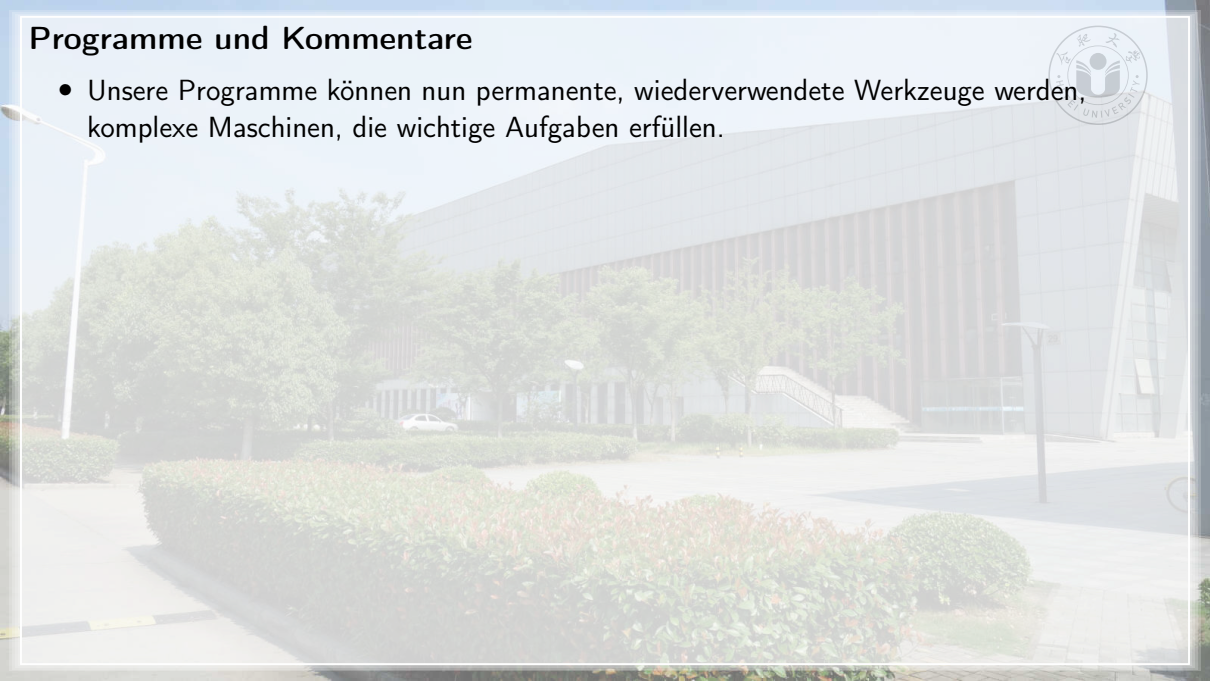
Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrer Schritte durchführen und aus mehreren Zeilen Code bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeichert, z. B. `hallo.py`.
- In genau dem Moment, in dem wir damit beginnen echte Programme zu schreiben, passieren zwei Dinge:
 1. Unser Code wird viel komplexer.
 2. Unser Code kann wieder verwendet werden, also mehrmals ausgeführt werden. Von uns, heute, morgen, oder in zehn Jahren. Und mit anderen Leuten geteilt werden, die den Code dann heute, morgen, oder in zehn Jahren ausführen.
- Das ändert die Qualität unseres Kodes extrem stark.
- Bisher war der Python-Interpreter im Grunde ein Taschenrechner für uns.
- Jetzt werden unsere Programme zu Werkzeugen, die wir hunderte Male verwenden oder Ziegel in riesigen Architekturen.

Programme und Kommentare

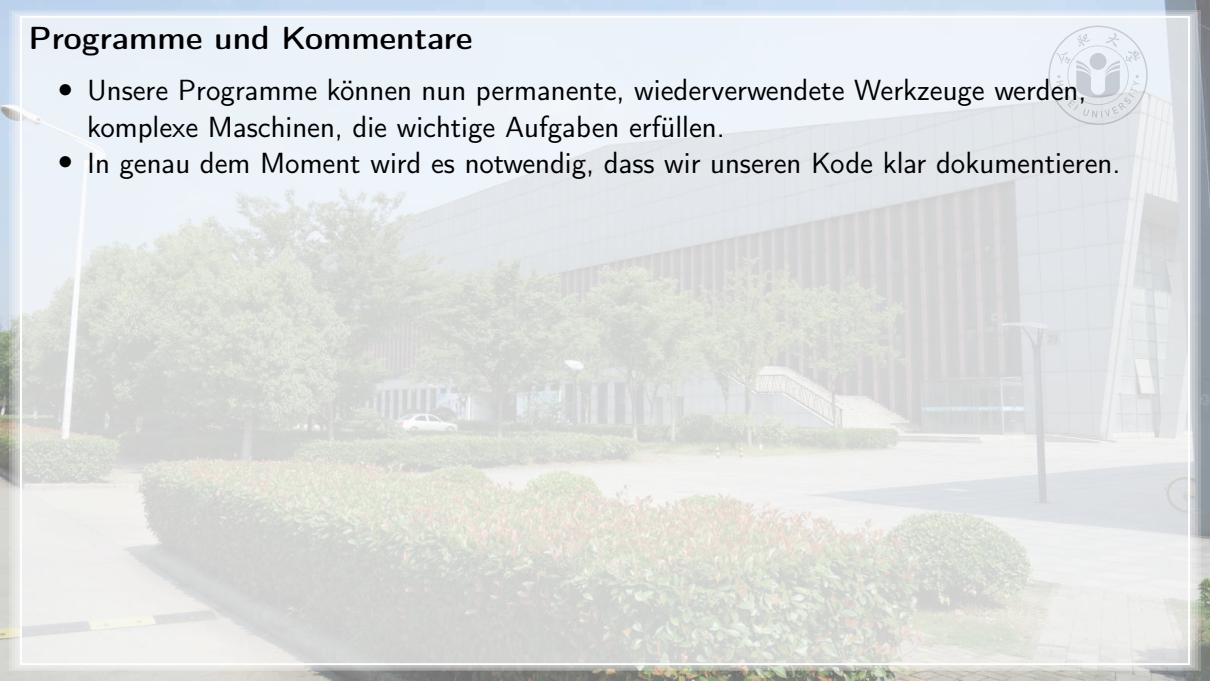
- Unsere Programme können nun permanente, wiederverwendete Werkzeuge werden, komplexe Maschinen, die wichtige Aufgaben erfüllen.



Programme und Kommentare



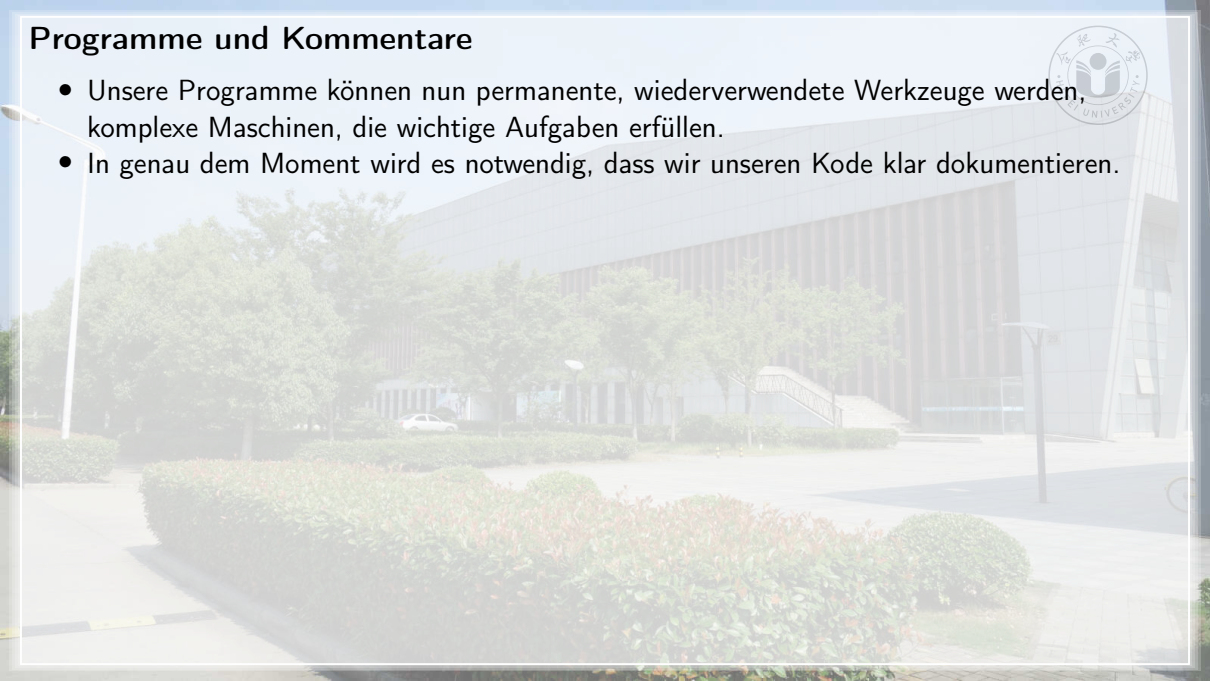
- Unsere Programme können nun permanente, wiederverwendete Werkzeuge werden, komplexe Maschinen, die wichtige Aufgaben erfüllen.
- In genau dem Moment wird es notwendig, dass wir unseren Code klar dokumentieren.



Programme und Kommentare



- Unsere Programme können nun permanente, wiederverwendete Werkzeuge werden, komplexe Maschinen, die wichtige Aufgaben erfüllen.
- In genau dem Moment wird es notwendig, dass wir unseren Code klar dokumentieren.



Programme und Kommentare



- Unsere Programme können nun permanente, wiederverwendete Werkzeuge werden, komplexe Maschinen, die wichtige Aufgaben erfüllen.
- In genau dem Moment wird es notwendig, dass wir unseren Code klar dokumentieren.
- Wir werden niemals nur Code schreiben – wir schreiben immer auch Kommentare, die beschreiben, was unser Code tut.
- Wir üben das von Anfang an. Wir **müssen** das konsequent durchziehen.

Programme und Kommentare



- Unsere Programme können nun permanente, wiederverwendete Werkzeuge werden, komplexe Maschinen, die wichtige Aufgaben erfüllen.
- In genau dem Moment wird es notwendig, dass wir unseren Code klar dokumentieren.
- Wir werden niemals nur Code schreiben – wir schreiben immer auch Kommentare, die beschreiben, was unser Code tut.
- Wir üben das von Anfang an. Wir **müssen** das konsequent durchziehen.

Gute Praxis

Kommentare helfen uns, zu beschreiben was der Code in Programmen tut und sind ein wichtiger Teil der Dokumentation unsere Codes. Kommentare beginnen mit dem Zeichen `#`. Der Python-Interpreter ignoriert allen Text nach diesem Zeichen bis zum Ende der Zeile. Kommentare können eine ganze Zeile einnehmen oder wir können zwei Leerzeichen nach einem Python-Kommando einfügen und dann einen Kommentar beginnen²⁰.

Programme und Kommentare



- Unsere Programme können nun permanente, wiederverwendete Werkzeuge werden, komplexe Maschinen, die wichtige Aufgaben erfüllen.
- In genau dem Moment wird es notwendig, dass wir unseren Code klar dokumentieren.
- Wir werden niemals nur Code schreiben – wir schreiben immer auch Kommentare, die beschreiben, was unser Code tut.
- Wir üben das von Anfang an. Wir **müssen** das konsequent durchziehen.

Gute Praxis

Kommentare helfen uns, zu beschreiben was der Code in Programmen tut und sind ein wichtiger Teil der Dokumentation unsere Codes. Kommentare beginnen mit dem Zeichen `#`. Der Python-Interpreter ignoriert allen Text nach diesem Zeichen bis zum Ende der Zeile. Kommentare können eine ganze Zeile einnehmen oder wir können zwei Leerzeichen nach einem Python-Kommando einfügen und dann einen Kommentar beginnen²⁰.

- In diesem Kurs werden wir immer grundlegende neue Elemente der Programmiersprache und wichtige Best Practices zusammen lernen.



Variablen Zuweisen



Variablen Zuweisen

- Schauen wir uns also ein Beispiel an.



Variablen Zuweisen



- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.

```
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 {int_var}.") # 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} = {int(new_var)}")
```

Variablen Zuweisen



- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.
- Dieses Programm macht nichts sinnvolles.

```
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 {int_var}.")  # 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} = {int_var}")
```

Variablen Zuweisen



- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.
- Dieses Programm macht nichts sinnvolles.
- Aber es zeigt einige Dinge, die wir mit Variablen machen können.

```
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 {int_var}.") # 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 = }.")
```

Variablen Zuweisen



- Es fängt damit an, den `int`-Wert `1` einer Variablen mit dem Namen `int_var` zuzuweisen.

```
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 {int_var}.") # 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} = {int_var}")
```

Variablen Zuweisen



- Es fängt damit an, den `int`-Wert `1` einer Variablen mit dem Namen `int_var` zuzuweisen.
- Wir hätten auch irgendeinen anderen Namen für die Variable nehmen können, z. B. `my_value`, `cow`, `race_car`, so lange er keine Sonderzeichen wie Leerzeichen oder Zeilenumbrüche beinhaltet.

```
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 {int_var}.") # 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} = {.}")
```

Variablen Zuweisen



- Aber wir haben eben `int_var` genommen.

```
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 {int_var}.") # 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 = }.")
```

Variablen Zuweisen



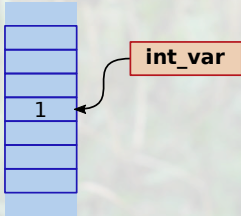
- Aber wir haben eben `int_var` genommen.
- Das `=` weist den Wert `1` der Variablen `int_value` zu.

```
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 {int_var}.") # 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} = ")."
```

Variablen Zuweisen



- Aber wir haben eben `int_var` genommen.
- Das `=` weist den Wert `1` der Variablen `int_value` zu.
- Der Wert `1` steht danach irgendwo im Speicher und `int_var` ist ein Name, der auf diese Speicherstelle zeigt.



Variablen Zuweisen



- Jetzt können wir `int_var` wie jeden anderen beliebigen Wert verwenden.

```
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 {int_var}.") # 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 = }.")
```

Variablen Zuweisen



- Jetzt können wir `int_var` wie jeden anderen beliebigen Wert verwenden.

```
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 {int_var}.") # 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} = {int(new_var)}")
```

Variablen Zuweisen



- Jetzt können wir `int_var` wie jeden anderen beliebigen Wert verwenden.
- Wir können `2 + int_var` berechnen und das Ergebnis der `print`-Funktion übergeben.

```
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 {int_var}.") # 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 = }.")
```

Variablen Zuweisen



- Jetzt können wir `int_var` wie jeden anderen beliebigen Wert verwenden.
- Wir können `2 + int_var` berechnen und das Ergebnis der `print`-Funktion übergeben.
- Diese druckt dann den Text `3` in den standard output stream (stdout) unseres Programms.

```
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 {int_var}.") # 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 = }.")
```

Variablen Zuweisen



- Wir können `int_var` auch in f-Strings verwenden.

```
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 {int_var}.") # 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} = {int_var}")
```

Variablen Zuweisen



- Wir können `int_var` auch in f-Strings verwenden.
- `f"int_var has value {int_var}."` wird dann zu `"int_var has value 1."` interpoliert.

```
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 {int_var}.") # 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} = {int_var}")
```

Variablen Zuweisen



- Variablen werden “Variablen” genannt und nicht “Konstanten”, weil wir ihnen auch neue Werte zuweisen können.

```
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 {int_var}.") # 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} = {.}")
```

Variablen Zuweisen



- Variablen werden “Variablen” genannt und nicht “Konstanten”, weil wir ihnen auch neue Werte zuweisen können.
- Wir können `int_var` also updaten und ihm einen neuen Wert zuweisen.

```
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 {int_var}.") # 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 = }.")
```

Variablen Zuweisen



- Variablen werden “Variablen” genannt und nicht “Konstanten”, weil wir ihnen auch neue Werte zuweisen können.
- Wir können `int_var` also updaten und ihm einen neuen Wert zuweisen.
- Wir können also `int_var = (3 * int_var) + 1` machen.

```
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 {int_var}.")  # 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} = {.}")
```

Variablen Zuweisen



- Wir können also `int_var = (3 * int_var) + 1` machen.
- In dieser Berechnung wird der alte Wert von `int_var` benutzt.

```
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 {int_var}.") # 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} = {.}")
```

Variablen Zuweisen



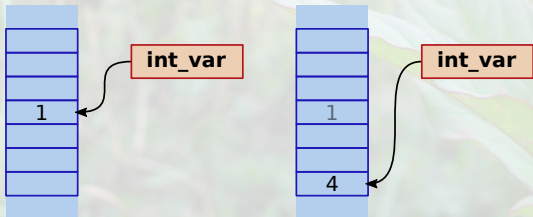
- Wir können also `int_var = (3 * int_var) + 1` machen.
- In dieser Berechnung wird der alte Wert von `int_var` benutzt.
- Wie berechnen also `(3 * 1) + 1`, was `4` ergibt.

```
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 {int_var}.") # 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} = {int(new_var)}")
```

Variablen Zuweisen



- Wie berechnen also $(3 * 1) + 1$, was 4 ergibt.
- Auch dieser Wert steht dann irgendwo im Speicher, und `int_var` zeigt darauf.



Variablen Zuweisen



- Der alte Wert `1` wird nicht mehr referenziert.

```
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 {int_var}.") # 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} = {int_var}")
```

Variablen Zuweisen



- Der alte Wert `1` wird nicht mehr referenziert.
- Der Python-Interpreter kann den entsprechenden Speicher freigeben und für etwas anderes benutzen.

```
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 {int_var}.") # 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} = {.}")
```

Variablen Zuweisen



- Der alte Wert `1` wird nicht mehr referenziert.
- Der Python-Interpreter kann den entsprechenden Speicher freigeben und für etwas anderes benutzen.
- Wenn wir jetzt nochmal `print(f"int_var is now {int_var}.")`, wird stattdessen `int_var is now 4.` auf dem stdout ausgegeben.

```
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 {int_var}.") # 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 = }.")
```

Variablen Zuweisen

- Natürlich können wir mehrere Variablen haben!



```
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 {int_var}.")  # 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} = {int_var}")
```

Variablen Zuweisen



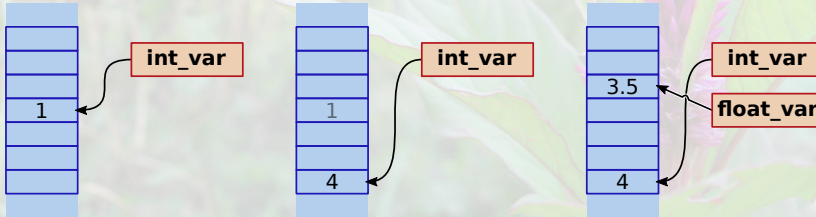
- Natürlich können wir mehrere Variablen haben!
- Das Kommando `float_var = 3.5` erstellt eine Variable namens `float_var`.

```
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 {int_var}.") # 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} = {int_var}")
```

Variablen Zuweisen



- Natürlich können wir mehrere Variablen haben!
- Das Kommando `float_var = 3.5` erstellt eine Variable namens `float_var`.
- Es allokiert den entsprechenden Speicher, schreibt den Fließkommawert `3.5` hinein, und lasse `float_var` darauf zeigen.



Variablen Zuweisen



- Wir können auch diese Variable in f-Strings verwenden.

```
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 {int_var}.") # 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} = {int(new_var)}")
```

Variablen Zuweisen



- Wir können auch diese Variable in f-Strings verwenden.
- `print(f"float_var has value {float_var}.")` wird zu `"float_var has value 3.5."` interpoliert.

```
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 {int_var}.") # 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} = {int(new_var)}")
```

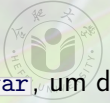
Variablen Zuweisen



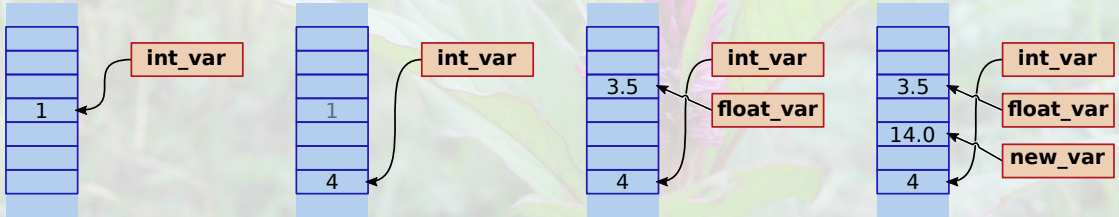
- In einem letzten Schritt erstellen wir eine dritte Variable mit dem Namen `new_var`, um das Ergebnis der Berechnung `new_var = float_var * int_var` zu speichern.

```
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 {int_var}.") # 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} = {int_var}")
```

Variablen Zuweisen



- In einem letzten Schritt erstellen wir eine dritte Variable mit dem Namen `new_var`, um das Ergebnis der Berechnung `new_var = float_var * int_var` zu speichern.
- Das ist das Ergebnis von `3.5 * 4`, also der `float`-Wert `14.0`.



Variablen Zuweisen



- Zu guter Letzt drucken wir noch `print(f"new_var = {new_var}.")`, was als `new_var = 14.0.` auf stdout erscheint.

```
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 {int_var}.")  # 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} = {new_var}.")
```

Variablen Zuweisen



- Zu guter Letzt drucken wir noch `print(f"new_var = {new_var}.")`, was als `new_var = 14.0.` auf stdout erscheint.
- Erinnern Sie sich noch an eine Möglichkeit, diese Ausgabe noch leichter zu bekommen?

```
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 {int_var}.") # 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} = {int_var}")
```

Variablen Zuweisen



- Schauen wir uns also ein Beispiel an.
- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.
- Schauen wir uns die gesamte Ausgabe auf den `stdout` unseres Programms an.

```
1 3
2 int_var has value 1.
3 int_var is now 4.
4 float_var has value 3.5.
5 new_var = 14.0.
```

Variablen Zuweisen



- Schauen wir uns also ein Beispiel an.
- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.
- Schauen wir uns die gesamte Ausgabe auf den `stdout` unseres Programms an.
- Das passt.

```
1 3
2 int_var has value 1.
3 int_var is now 4.
4 float_var has value 3.5.
5 new_var = 14.0.
```



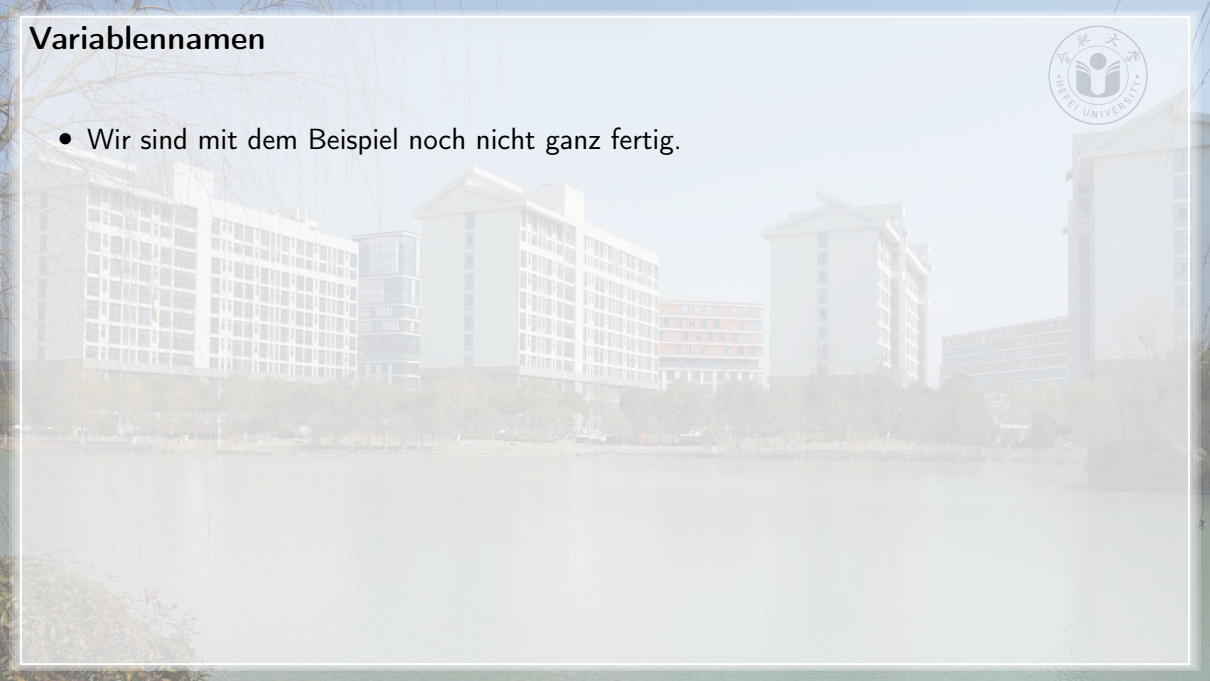
Kode und Stil



Variablennamen



- Wir sind mit dem Beispiel noch nicht ganz fertig.



Variablennamen



- Wir sind mit dem Beispiel noch nicht ganz fertig.
- Ist Ihnen aufgefallen, wie wir die Variablen benannt haben?

Variablennamen



- Wir sind mit dem Beispiel noch nicht ganz fertig.
- Ist Ihnen aufgefallen, wie wir die Variablen benannt haben?
- In Kleinbuchstaben. Wir haben keine Großbuchstaben verwendet.

Variablennamen



- Wir sind mit dem Beispiel noch nicht ganz fertig.
- Ist Ihnen aufgefallen, wie wir die Variablen benannt haben?
- In Kleinbuchstaben. Wir haben keine Großbuchstaben verwendet.
- Das ist der De-facto-Standard in Python.

Variablennamen



- Wir sind mit dem Beispiel noch nicht ganz fertig.
- Ist Ihnen aufgefallen, wie wir die Variablen benannt haben?
- In Kleinbuchstaben. Wir haben keine Großbuchstaben verwendet.
- Das ist der De-facto-Standard in Python:

Gute Praxis

Variablennamen sollen in Kleinbuchstaben geschrieben werden. Wörter werden durch Unterstriche getrennt²⁰.

Variablennamen



- Wir sind mit dem Beispiel noch nicht ganz fertig.
- Ist Ihnen aufgefallen, wie wir die Variablen benannt haben?
- In Kleinbuchstaben. Wir haben keine Großbuchstaben verwendet.
- Das ist der De-facto-Standard in Python:

Gute Praxis

Variablennamen sollen in Kleinbuchstaben geschrieben werden. Wörter werden durch Unterstriche getrennt²⁰.

- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .

Kode und Stil



- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .

Kode und Stil



- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .
- Warum denken wir über sowas nach, wo wir doch gerade Programmieren lernen?

Kode und Stil



- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .
- Warum denken wir über sowas nach, wo wir doch gerade Programmieren lernen?
- Warum ist das wichtig?

Kode und Stil



- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .
- Warum denken wir über sowas nach, wo wir doch gerade Programmieren lernen?
- Warum ist das wichtig? Warum ist das wichtig, dass wir das **jetzt** gleich lernen?
- Weil das befolgen von *Best Practices* nichts ist, dass man nachträglich später machen kann.

Kode und Stil



- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .
- Warum denken wir über sowas nach, wo wir doch gerade Programmieren lernen?
- Warum ist das wichtig? Warum ist das wichtig, dass wir das **jetzt** gleich lernen?
- Weil das befolgen von *Best Practices* nichts ist, dass man nachträglich später machen kann.
- Sie werden **niemals** die Zeit haben, den Stil Ihres alten Codes zu verbessern.

Kode und Stil



- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .
- Warum denken wir über sowas nach, wo wir doch gerade Programmieren lernen?
- Warum ist das wichtig? Warum ist das wichtig, dass wir das **jetzt** gleich lernen?
- Weil das befolgen von *Best Practices* nichts ist, dass man nachträglich später machen kann.
- Sie werden **niemals** die Zeit haben, den Stil Ihres alten Codes zu verbessern.
- Das ist auch nichts, dass man einfach so anfangen kann zu machen.

Kode und Stil



- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .
- Warum denken wir über sowas nach, wo wir doch gerade Programmieren lernen?
- Warum ist das wichtig? Warum ist das wichtig, dass wir das **jetzt** gleich lernen?
- Weil das befolgen von *Best Practices* nichts ist, dass man nachträglich später machen kann.
- Sie werden **niemals** die Zeit haben, den Stil Ihres alten Codes zu verbessern.
- Das ist auch nichts, dass man einfach so anfangen kann zu machen.
- Wenn Sie gelernt haben, eine Sache auf eine bestimmte Art zu machen, dann ist es immer schwer, auf eine andere Art umzuschalten.

Kode und Stil



- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .
- Warum denken wir über sowas nach, wo wir doch gerade Programmieren lernen?
- Warum ist das wichtig? Warum ist das wichtig, dass wir das **jetzt** gleich lernen?
- Weil das befolgen von *Best Practices* nichts ist, dass man nachträglich später machen kann.
- Sie werden **niemals** die Zeit haben, den Stil Ihres alten Codes zu verbessern.
- Das ist auch nichts, dass man einfach so anfangen kann zu machen.
- Wenn Sie gelernt haben, eine Sache auf eine bestimmte Art zu machen, dann ist es immer schwer, auf eine andere Art umzuschalten.
- Wenn ein Koch-Azubi nicht beigebracht bekommt, sich vor dem Essenmachen die Hände zu waschen, dann wird er es später auch nicht von sich aus konsistent machen, auch dann nicht, wenn es ihm einmal explizit gesagt wird.

Kode und Stil



- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .
- Warum denken wir über sowas nach, wo wir doch gerade Programmieren lernen?
- Warum ist das wichtig? Warum ist das wichtig, dass wir das **jetzt** gleich lernen?
- Weil das befolgen von *Best Practices* nichts ist, dass man nachträglich später machen kann.
- Sie werden **niemals** die Zeit haben, den Stil Ihres alten Codes zu verbessern.
- Das ist auch nichts, dass man einfach so anfangen kann zu machen.
- Wenn Sie gelernt haben, eine Sache auf eine bestimmte Art zu machen, dann ist es immer schwer, auf eine andere Art umzuschalten.
- Wenn ein Koch-Azubi nicht beigebracht bekommt, sich vor dem Essenmachen die Hände zu waschen, dann wird er es später auch nicht von sich aus konsistent machen, auch dann nicht, wenn es ihm einmal explizit gesagt wird.
- Das befolgen von Stilrichtlinien und Best Practices ist eine Angewohnheit.

Kode und Stil



- Das ist wieder so ein komischer Stilhinweis, wie das mit den Kommentaren und den doppelten Anführungszeichen für mehrzeilige Strings. . .
- Warum denken wir über sowas nach, wo wir doch gerade Programmieren lernen?
- Warum ist das wichtig? Warum ist das wichtig, dass wir das **jetzt** gleich lernen?
- Weil das befolgen von *Best Practices* nichts ist, dass man nachträglich später machen kann.
- Sie werden **niemals** die Zeit haben, den Stil Ihres alten Codes zu verbessern.
- Das ist auch nichts, dass man einfach so anfangen kann zu machen.
- Wenn Sie gelernt haben, eine Sache auf eine bestimmte Art zu machen, dann ist es immer schwer, auf eine andere Art umzuschalten.
- Wenn ein Koch-Azubi nicht beigebracht bekommt, sich vor dem Essenmachen die Hände zu waschen, dann wird er es später auch nicht von sich aus konsistent machen, auch dann nicht, wenn es ihm einmal explizit gesagt wird.
- Das befolgen von Stilrichtlinien und Best Practices ist eine Angewohnheit.
- Und die lernen wir hier gleich mit.

PEP8

- Für viele Programmiersprachen gibt es umfangreiche und klare Stilrichtlinien.



PEP8



- Für viele Programmiersprachen gibt es umfangreiche und klare Stilrichtlinien.
- Weil wir meistens kollaborativ an größeren Projekten arbeiten, ist es wichtig, Kode in einem konsistenten Stil zu schreiben.

PEP8



- Für viele Programmiersprachen gibt es umfangreiche und klare Stilrichtlinien.
- Weil wir meistens kollaborativ an größeren Projekten arbeiten, ist es wichtig, Kode in einem konsistenten Stil zu schreiben.
- Alle Mitarbeiter sollen allen Quellcode leicht lesen und verstehen können.

PEP8



- Für viele Programmiersprachen gibt es umfangreiche und klare Stilrichtlinien.
- Weil wir meistens kollaborativ an größeren Projekten arbeiten, ist es wichtig, Kode in einem konsistenten Stil zu schreiben.
- Alle Mitarbeiter sollen allen Quellcode leicht lesen und verstehen können.
- Wenn jeder Kode in einem anderen Stil schreibt, vielleicht andere Einrückungen und Namenskonventionen verwendet, dann wird das viel schwerer und verwirrender.

PEP8



- Für viele Programmiersprachen gibt es umfangreiche und klare Stilrichtlinien.
- Weil wir meistens kollaborativ an größeren Projekten arbeiten, ist es wichtig, Kode in einem konsistenten Stil zu schreiben.
- Alle Mitarbeiter sollen allen Quellcode leicht lesen und verstehen können.
- Wenn jeder Kode in einem anderen Stil schreibt, vielleicht andere Einrückungen und Namenskonventionen verwendet, dann wird das viel schwerer und verwirrender.
- Daher sagen uns Stilrichtlinien, wie wir Dinge benennen und unseren Kode formatieren sollen.

PEP8



- Für viele Programmiersprachen gibt es umfangreiche und klare Stilrichtlinien.
- Weil wir meistens kollaborativ an größeren Projekten arbeiten, ist es wichtig, Kode in einem konsistenten Stil zu schreiben.
- Alle Mitarbeiter sollen allen Quellcode leicht lesen und verstehen können.
- Wenn jeder Kode in einem anderen Stil schreibt, vielleicht andere Einrückungen und Namenskonventionen verwendet, dann wird das viel schwerer und verwirrender.
- Daher sagen uns Stilrichtlinien, wie wir Dinge benennen und unseren Kode formatieren sollen.

Gute Praxis

Die wichtigsten Stilrichtlinien für die Programmiersprache Python ist PEP8: *Style Guide for Python Code*²⁰, die wir unter <https://peps.python.org/pep-0008> finden können. Python-Kode der PEP8 verletzt ist falscher Python-Kode.



Zusammenfassung



Zusammenfassung



- Fertig.





谢谢您门！
Thank you!
Vielen Dank!



References I



- [1] Daniel J. Barrett. *Efficient Linux at the Command Line*. Sebastopol, CA, USA: O'Reilly Media, Inc., Feb. 2022. ISBN: 978-1-0981-1340-7 (siehe S. 101, 102).
- [2] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 101).
- [3] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 101).
- [4] Florian Bruhin. *Python f-Strings*. Winterthur, Switzerland: Bruhin Software, 31. Mai 2023. URL: <https://fstring.help> (besucht am 2024-07-25) (siehe S. 101).
- [5] David Clinton und Christopher Negus. *Ubuntu Linux Bible*. 10. Aufl. Bible Series. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 10. Nov. 2020. ISBN: 978-1-119-72233-5 (siehe S. 102).
- [6] "Formatted String Literals". In: *Python 3 Documentation. The Python Tutorial*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. Kap. 7.1.1. URL: <https://docs.python.org/3/tutorial/inputoutput.html#formatted-string-literals> (besucht am 2024-07-25) (siehe S. 101).
- [7] Bhavesh Gawade. "Mastering F-Strings in Python: Efficient String Handling in Python Using Smart F-Strings". In: *C O D E B*. Mumbai, Maharashtra, India: Code B Solutions Pvt Ltd, 25. Apr.–3. Juni 2025. URL: <https://code-b.dev/blog/f-strings-in-python> (besucht am 2025-08-04) (siehe S. 101).
- [8] Olaf Górski. "Why f-strings are awesome: Performance of different string concatenation methods in Python". In: *DEV Community*. Sacramento, CA, USA: DEV Community Inc., 8. Nov. 2022. URL: <https://dev.to/grski/performance-of-different-string-concatenation-methods-in-python-why-f-strings-are-awesome-2e97> (besucht am 2025-08-04) (siehe S. 101).
- [9] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 101).
- [10] Matthew Helmke. *Ubuntu Linux Unleashed 2021 Edition*. 14. Aufl. Reading, MA, USA: Addison-Wesley Professional, Aug. 2020. ISBN: 978-0-13-668539-5 (siehe S. 102).

References II



- [11] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 101).
- [12] „stderr, stdin, stdout – Standard I/O Streams“. In: *POSIX.1-2024: The Open Group Base Specifications Issue 8, IEEE Std 1003.1™-2024 Edition*. Hrsg. von Andrew Josey. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE) und San Francisco, CA, USA: The Open Group, 8. Aug. 2024. URL: <https://pubs.opengroup.org/onlinepubs/9799919799/functions/stdin.html> (besucht am 2024-10-30) (siehe S. 101, 102).
- [13] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 101).
- [14] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 101).
- [15] Aaron Maxwell. *What are f-strings in Python and how can I use them?* Oakville, ON, Canada: Infinite Skills Inc, Juni 2017. ISBN: 978-1-4919-9486-3 (siehe S. 101).
- [16] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 101).
- [17] Ellen Siever, Stephen Figgins, Robert Love und Arnold Robbins. *Linux in a Nutshell*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2009. ISBN: 978-0-596-15448-6 (siehe S. 101).
- [18] Eric V. „[ericvsmith](#)“ Smith. *Literal String Interpolation*. Python Enhancement Proposal (PEP) 498. Beaverton, OR, USA: Python Software Foundation (PSF), 6. Nov. 2016–9. Sep. 2023. URL: <https://peps.python.org/pep-0498> (besucht am 2024-07-25) (siehe S. 101).
- [19] Linus Torvalds. „The Linux Edge“. *Communications of the ACM (CACM)* 42(4):38–39, Apr. 1999. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/299157.299165 (siehe S. 101).
- [20] Guido van Rossum, Barry Warsaw und Alyssa Coghlan. *Style Guide for Python Code*. Python Enhancement Proposal (PEP) 8. Beaverton, OR, USA: Python Software Foundation (PSF), 5. Juli 2001. URL: <https://peps.python.org/pep-0008> (besucht am 2024-07-27) (siehe S. 30–35, 73–78, 89–94).

References III



- [21] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: **978-0-13-792931-3** (siehe S. 101).
- [22] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: <https://thomasweise.github.io/programmingWithPython> (besucht am 2025-01-05) (siehe S. 101).
- [23] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: **978-1-78439-687-9** (siehe S. 101).

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^{3,16,23}. Learn more at <https://www.gnu.org/software/bash>.

f-string let you include the results of expressions in strings^{4,6–8,15,18}. They can contain expressions (in curly braces) like `f"a{6-1}b"` that are then transformed to text via (string) interpolation, which turns the string to `"a5b"`. F-strings are delimited by `f"..."`.

Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows^{1,9,17,19,21}. 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.

Microsoft Windows is a commercial proprietary operating system². 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>.

Python The Python programming language^{11,13,14,22}, i.e., what you will learn about in our book²². Learn more at <https://python.org>.

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 stdout)¹². 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)¹². 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.

Glossary (in English) II



stdout The *standard output stream* is one of the three pre-defined streams of a console process (together with the `stdin` and the `stderr`)¹². 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.

(string) interpolation In Python, string interpolation is the process where all the expressions in an f-string are evaluated and the final string is constructed. An example for string interpolation is turning `f"Rounded {1.234:.2f}"` to `"Rounded 1.23"`.

terminal A terminal is a text-based window where you can enter commands and execute them^{1,5}. 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 `Win`+`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^{5,10}. 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>.