



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-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

1. Einleitung
2. Variablen und Kommentare
3. Variablen Zuweisen
4. Kode und Stil
5. LIU Hui's Methode, π zu Approximieren
6. Mehrfachzuweisungen und Wertetausch
7. 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 einer Zeile Kode 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 einer Zeile Kode 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 einer Zeile Kode 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

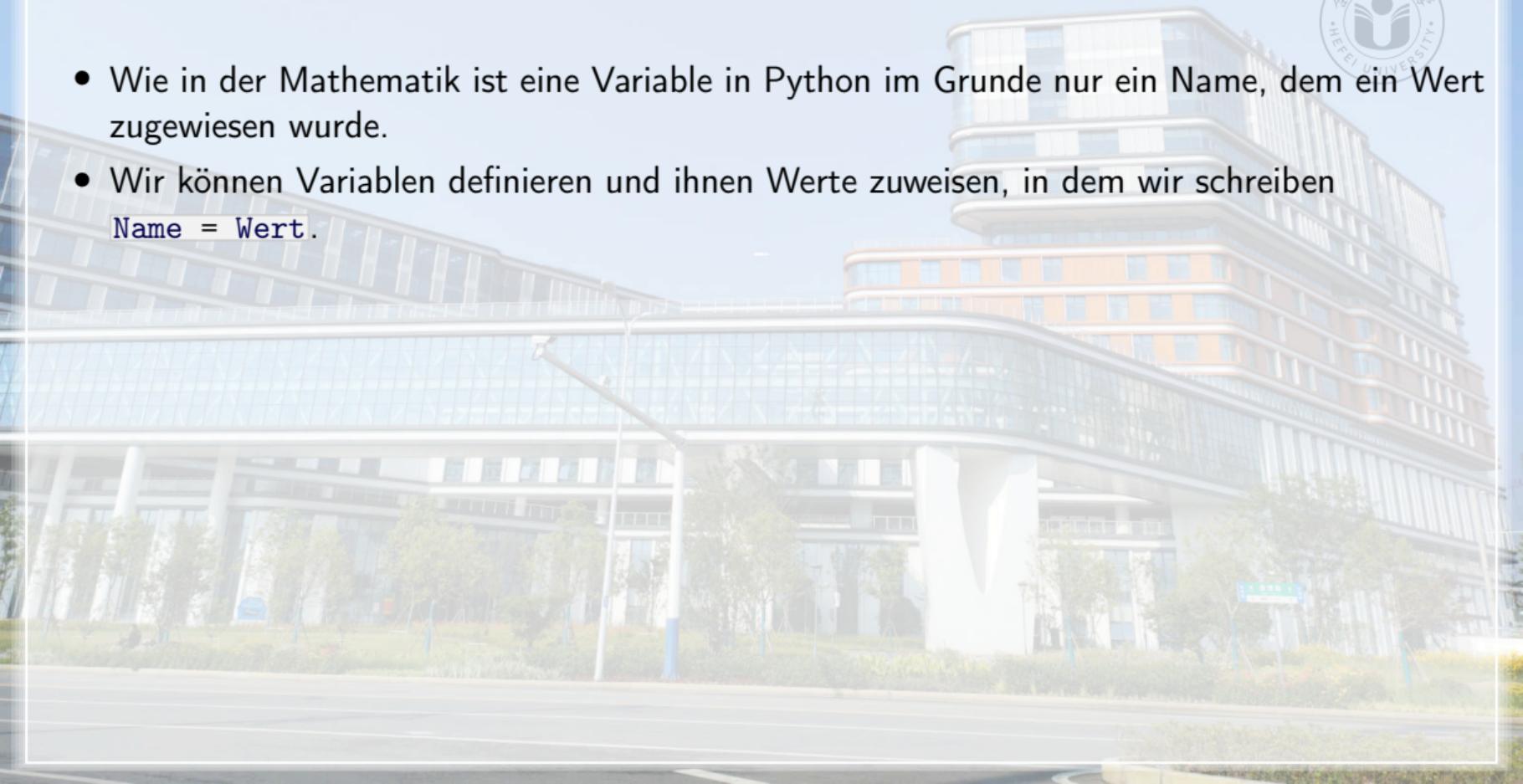


- Wie in der Mathematik ist eine Variable in Python im Grunde nur ein Name, dem ein Wert zugewiesen wurde.

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 mehrere 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 mehrere Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeicher, z. B. `hallo.py`.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrere Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeicher, 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 mehrere Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeicher, 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.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrere Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeicher, 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.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrere Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeicher, 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.

Programme



- Wir können nun Zwischenergebnisse speichern und wieder abrufen.
- Zum ersten Mal können wir nun Programme schreiben, die Berechnung über mehrere Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeicher, 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 mehrere Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeicher, 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 mehrere Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeicher, 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.
- 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 mehrere Schritte durchführen und aus mehreren Zeilen Kode bestehen.
- Python-Programme werden in Textdateien mit der Endung `.py` gespeicher, 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.
- 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 Kode 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 Kode 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 Kode klar dokumentieren.
- Wir werden niemals nur Kode schreiben – wir schreiben immer auch Kommentare, die beschreiben, was unser Kode 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 Kode klar dokumentieren.
- Wir werden niemals nur Kode schreiben – wir schreiben immer auch Kommentare, die beschreiben, was unser Kode tut.
- Wir üben das von Anfang an. Wir **müssen** das konsequent durchziehen.

Gute Praxis

Kommentare helfen uns, zu beschreiben was der Kode in Programmen tut und sind ein wichtiger Teil der Dokumentation unserer Kodes. 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 beginnen⁵⁹.



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 Kode klar dokumentieren.
- Wir werden niemals nur Kode schreiben – wir schreiben immer auch Kommentare, die beschreiben, was unser Kode tut.
- Wir üben das von Anfang an. Wir **müssen** das konsequent durchziehen.

Gute Praxis

Kommentare helfen uns, zu beschreiben was der Kode in Programmen tut und sind ein wichtiger Teil der Dokumentation unserer Kodes. 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 beginnen⁵⁹.

- 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 = }.')
```

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} = }).'
```

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} = })."
```

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



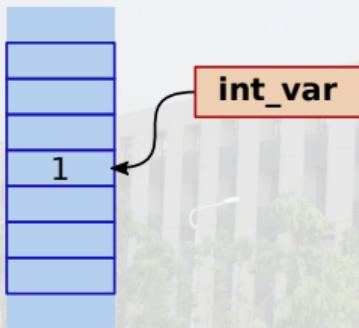
- Aber wir haben eben `int_var` genommen.
- Das `=` weist den Wert `1` der Variablen `int_value` zu.

```
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.
- 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} = )."
```

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} = )."
```

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 update 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 update 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



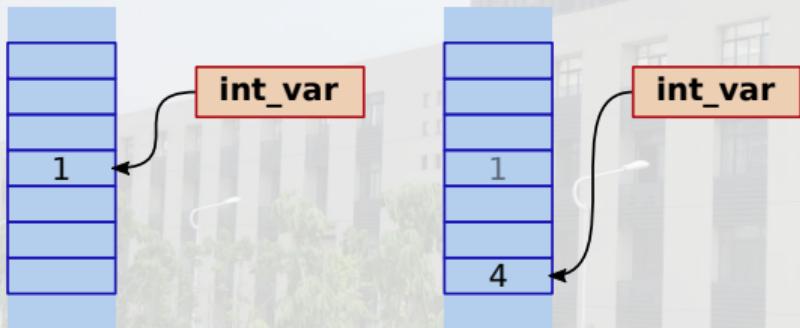
- Wir können also `int_var = (3 * int_var)+ 1` machen.
- In dieser Berechnung wird der alte Wert von `int_var` benutzt.
- Wie berechnen also `(3 * 1)+ 1`, was `4` ergibt.

```
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} = )."
```



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} = }).'
```

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} = })."
```

Variablen Zuweisen

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

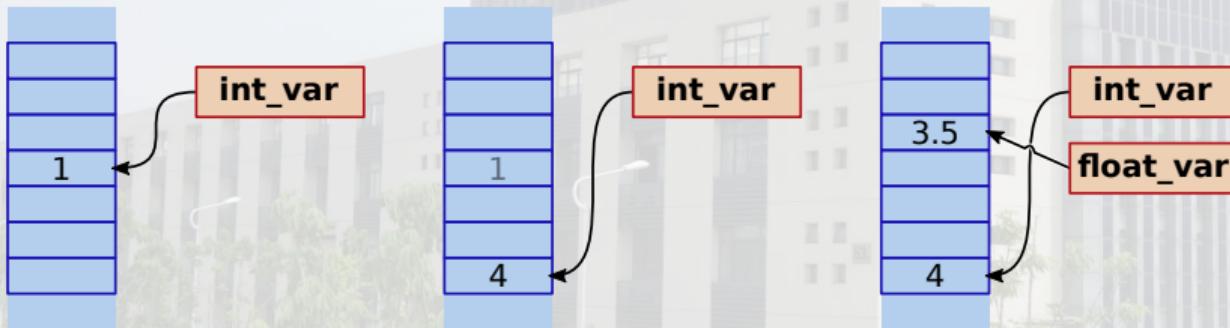
```
1 # We define a variable named "int_var" and assign the int value 1 to it.
2 int_var = 1
3
4 # We can use the variable int_var in computations like any other value.
5 print(2 + int_var) # This should print 2 + int_var = 2 + 1 = 3.
6
7 # We can also use the variable in f-strings.
8 print(f'int_var 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 = }.')
```





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 lässt `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} = }).'
```

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 = }).'
```

Variablen Zuweisen



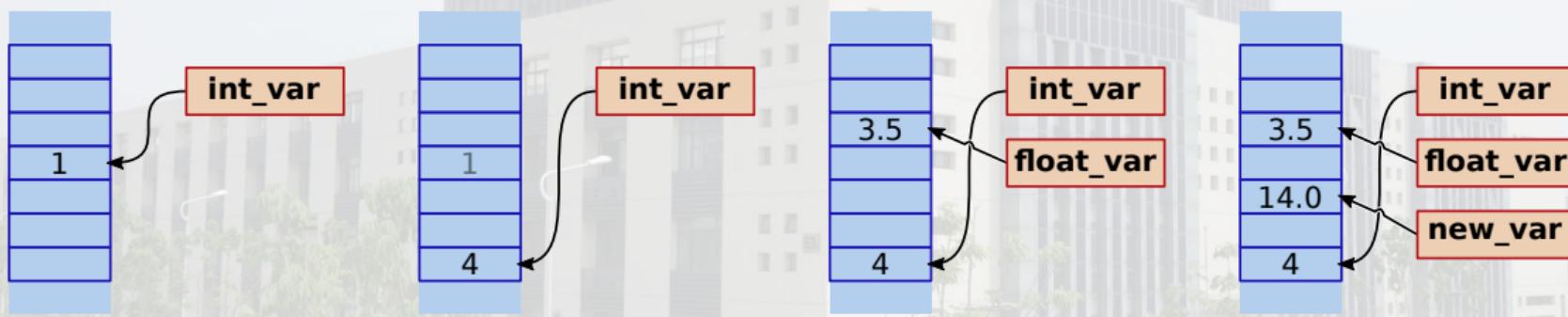
- In einem letzten Schritt erstellen wir eine dritte Variable mit dem Namen `new_var`, um das Ergebnis der Berechnung `new_var = float_var * int_var` zu speichern.

```
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} = )."
```

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} .')
```

Variablen Zuweisen



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

Variablen Zuweisen



- Schauen wir uns also ein Beispiel an.
- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.
- Nun führen wir das Program im Terminal aus.

Variablen Zuweisen



- Schauen wir uns also ein Beispiel an.
- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.
- Nun führen wir das Program im Terminal aus.
- Unter Ubuntu Linux können wir ein Terminal durch Druck auf **Ctrl**+**Alt**+**T** öffnen, unter Microsoft Windows durch Druck auf **Windows**+**R**, dann Schreiben von `cmd`, dann Druck auf **Enter**.

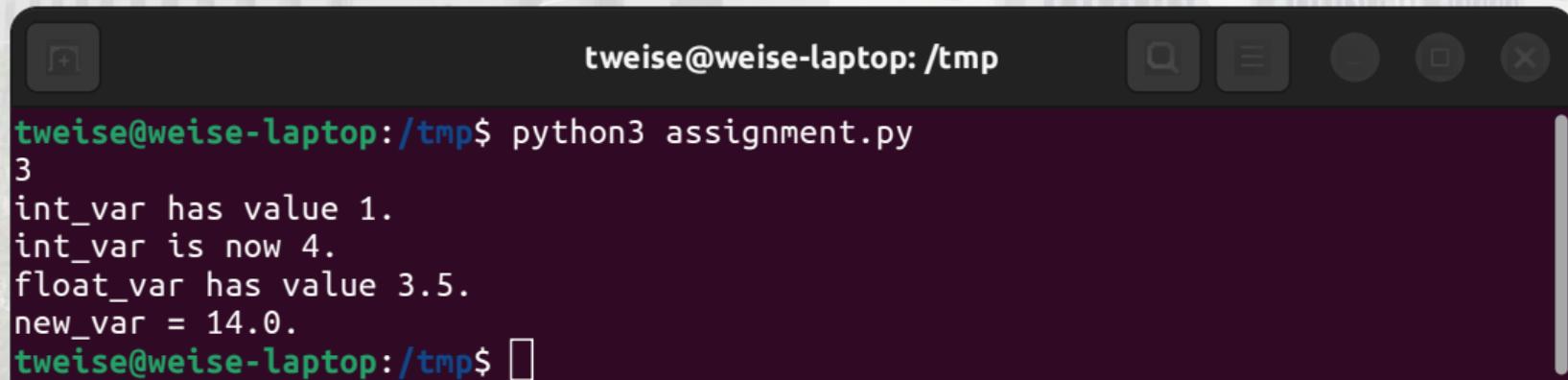
A screenshot of a terminal window titled "tweise@weise-laptop: /tmp". The window shows the command "python3 assignment.py" being run, followed by the output of the program which prints five variable assignments. The terminal has a dark theme with light-colored text and icons.

```
tweise@weise-laptop: /tmp$ python3 assignment.py
3
int_var has value 1.
int_var is now 4.
float_var has value 3.5.
new_var = 14.0.
tweise@weise-laptop: /tmp$
```

Variablen Zuweisen



- Schauen wir uns also ein Beispiel an.
- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.
- Unter Ubuntu Linux können wir ein Terminal durch Druck auf **Ctrl** + **Alt** + **T** öffnen, unter Microsoft Windows durch Druck auf **Windows** + **R**, dann Schreiben von `cmd`, dann Druck auf **Enter**.
- Auf beiden Betriebssystemen gehen wir mit Hilfe des Kommandos `cd` in den Ordner mit der Programmdatei `assignment.py`.



A screenshot of a terminal window titled "tweise@weise-laptop: /tmp". The window has standard OS X-style controls at the top right. The terminal displays the following output:

```
tweise@weise-laptop:~/tmp$ python3 assignment.py
3
int_var has value 1.
int_var is now 4.
float_var has value 3.5.
new_var = 14.0.
tweise@weise-laptop:~/tmp$
```

Variablen Zuweisen



- Schauen wir uns also ein Beispiel an.
- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.
- Auf beiden Betriebssystemen gehen wir mit Hilfe des Kommandos `cd` in den Ordner mit der Programmdatei `assignment.py`.
- Dort können wir diese dann durch `python3 assignment.py` ausführen.

A screenshot of a terminal window titled "tweise@weise-laptop: /tmp". The window has a dark background and light-colored text. At the top, there are several icons: a file folder, a magnifying glass, a list icon, a circle, a square, and a close button. The terminal output shows the following:

```
tweise@weise-laptop:~/tmp$ python3 assignment.py
3
int_var has value 1.
int_var is now 4.
float_var has value 3.5.
new_var = 14.0.
tweise@weise-laptop:~/tmp$
```

The text is in white, except for the command prompt which is in green.



Variablen Zuweisen

- Schauen wir uns also ein Beispiel an.
- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.
- Dort können wir diese dann durch `python3 assignment.py` ausführen.
- Die Ausgabe ist natürlich die selbe.

```
tweise@weise-laptop: /tmp
tweise@weise-laptop: /tmp$ python3 assignment.py
3
int_var has value 1.
int_var is now 4.
float_var has value 3.5.
new_var = 14.0.
tweise@weise-laptop: /tmp$
```



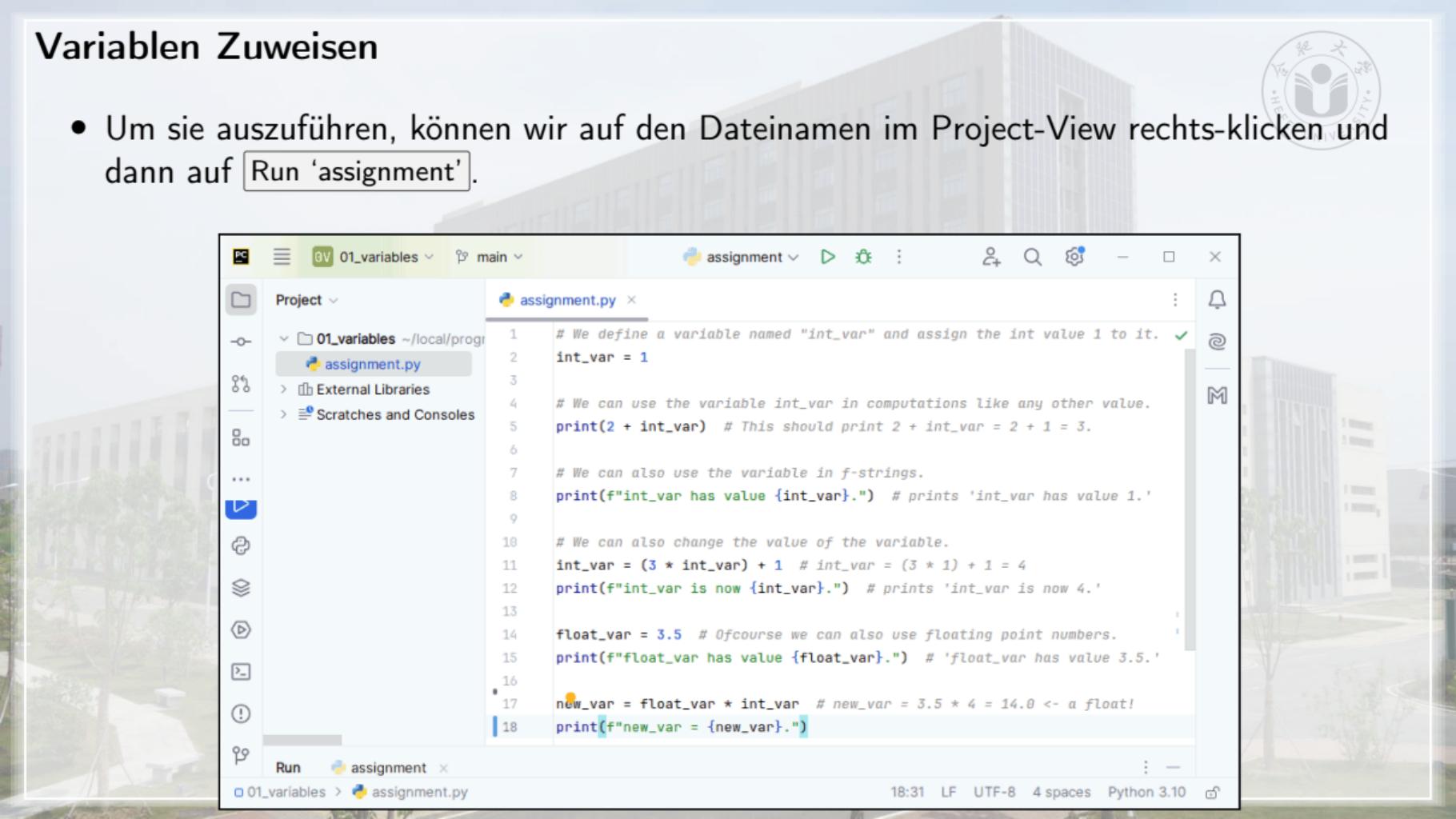
Variablen Zuweisen

- Schauen wir uns also ein Beispiel an.
- Hier ist ein Programm, das wir in der Datei `assignment.py` gespeichert haben.
- Alternativ können wir die Programmdatei auch in PyCharm dem Integrated Development Environment (IDE) öffnen.

```
tweise@weise-laptop: /tmp
tweise@weise-laptop: /tmp$ python3 assignment.py
3
int_var has value 1.
int_var is now 4.
float_var has value 3.5.
new_var = 14.0.
tweise@weise-laptop: /tmp$
```

Variablen Zuweisen

- Um sie auszuführen, können wir auf den Dateinamen im Project-View rechts-klicken und dann auf Run 'assignment'.



A screenshot of a Python code editor interface. The left sidebar shows a project structure with a folder '01_variables' containing a file 'assignment.py'. The main editor window displays the following Python code:

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

The code editor includes standard toolbars and status bars at the bottom showing the file path '01_variables > assignment.py', the current time '18:31', and encoding information 'LF UTF-8 4 spaces Python 3.10'.

Variablen Zuweisen



- Oder wir drücken einfach **Ctrl**+**↑**+**F10** im Editor.

The screenshot shows the PyCharm IDE interface. On the left is the Project tool window with a file named 'assignment.py' selected. In the center is the code editor with the following Python code:

```
assignment.py
# Create a variable named "int_var" and assign the int value 1 to it.
int_var = 1
# Now we can use int_var in computations like any other value.
# This should print 2 + int_var = 2 + 1 = 3.
print(2 + int_var)

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

# We can print the value of the variable.
print(int_var)

# We can also multiply int_var by another integer.
print((int_var) * 3) # int_var = (3 * 1) + 1 = 4
# Now int_var is 4.
print(int_var)

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

# int_var # new_var = 3.5 * 4 = 14.0 <- a float!
# print(new_var)

# We can also use float_var in computations.
print(int_var + float_var)
```

A context menu is open over the line 'int_var = 1', showing options like 'New', 'Cut', 'Copy', 'Paste', 'Find Usages', 'Inspect Code...', 'Refactor', 'Bookmarks', 'Reformat Code', 'Optimize Imports', 'Delete...', and 'Override File Type'. The 'Run 'assignment'' option is highlighted at the bottom of the menu.

Variablen Zuweisen

- So oder so wird das Programm ausgeführt und wir bekommen wieder genau die erwartete Ausgabe.

The screenshot shows a Python code editor interface with the following details:

- Project:** 01_variables
- File:** assignment.py
- Code Content:**

```
    # Create a variable named "int_var" and assign the int value 1 to it. ✓
int_var = 1
# This should print 2 + int_var = 2 + 1 = 3.

# Print the variable in f-strings.
print(f"Value {int_var}.") # prints 'int_var has value 1.'

# Print the value of the variable.
print(int_var) # int_var = (3 * 1) + 1 = 4
print(f"Value {int_var}.") # prints 'int_var is now 4.'

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

# int_var # new_var = 3.5 * 4 = 14.0 <- a float!
# print(new_var)."
```
- Contextual Menu (Open at int_var):**
 - New
 - Cut (Ctrl+X)
 - Copy (Ctrl+C)
 - Copy Path/Reference...
 - Paste (Ctrl+V)
 - Find Usages (Alt+Shift+F7)
 - Inspect Code...
 - Refactor
 - Bookmarks
 - Reformat Code (Ctrl+Alt+L)
 - Optimize Imports (Ctrl+Alt+O)
 - Delete...
 - Override File Type
- Bottom Bar:**
 - Run assignment (Ctrl+Shift+F10)
 - Debug assignment
 - 01_variables > assignment.py
 - 18:31 LF UTF-8 4 spaces Python 3.10



Variablen Zuweisen



- So oder so wird das Programm ausgeführt und wir bekommen wieder genau die erwartete Ausgabe.

The screenshot shows a Python IDE interface with the following details:

- Project:** 01_variables
- File:** assignment.py
- Code Content:**

```
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}.")
19
```
- Run Tab:** Shows the command: /usr/bin/python3.10 /home/tweisse/local/programming/python/programmingWithPythonCode/01_variables/assignment.py
- Output:**

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

Process finished with exit code 0
```
- Status Bar:** 01_variables > assignment.py, 18:31, LF, UTF-8, 4 spaces, Python 3.10



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 getrennt⁵⁹.



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

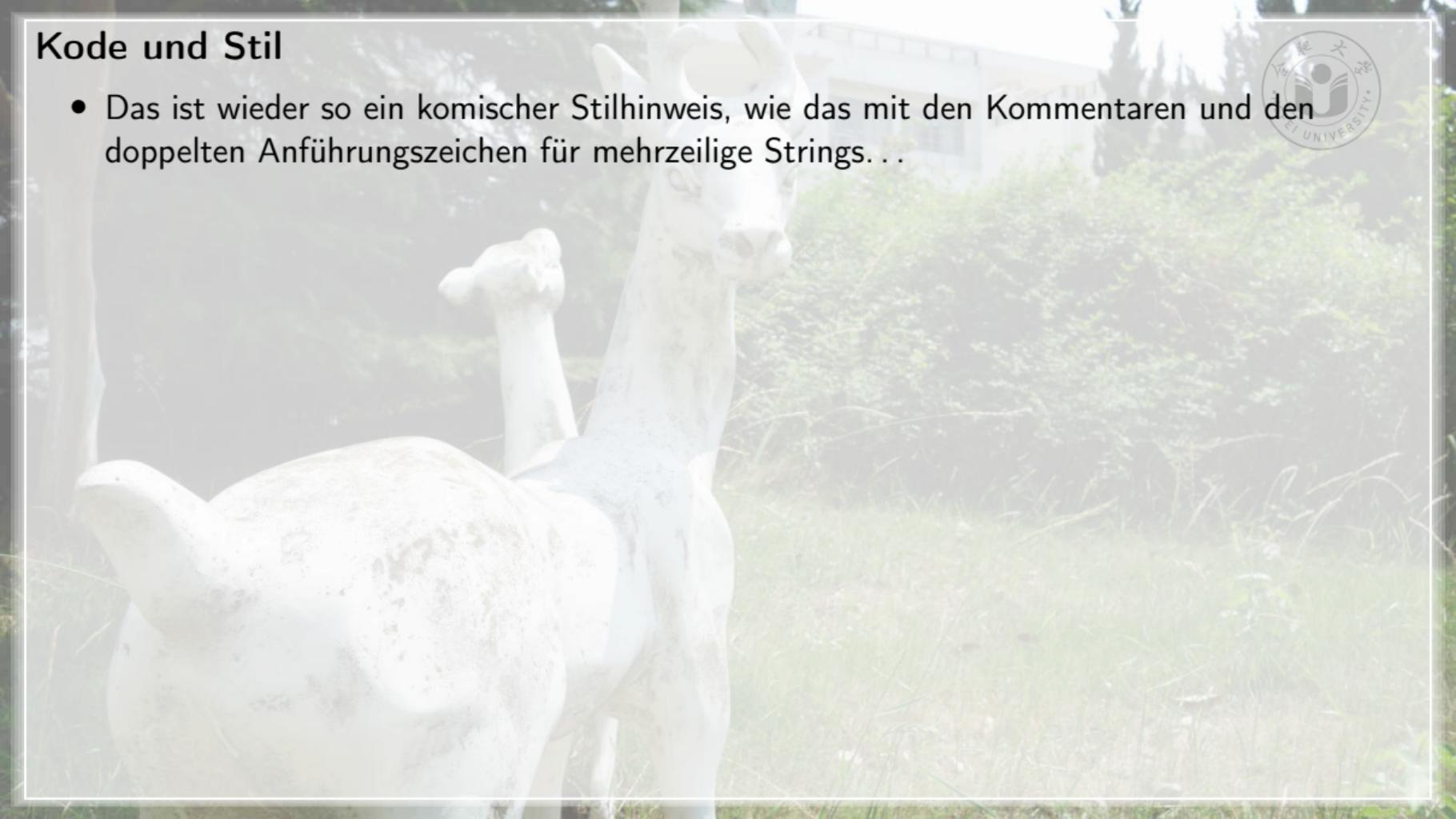
Variablennamen sollen in Kleinbuchstaben geschrieben werden. Wörter werden durch Unterstriche getrennt⁵⁹.

- 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 Kodes 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 Kodes 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 Kodes 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 Kodes 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 konsisten 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 Kodes 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 konsisten 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 Kodes 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 konsisten 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 konsisten 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 konsisten Stil zu schreiben.
- Alle Mitarbeiter sollen allen Quellkode 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 konsisten Stil zu schreiben.
- Alle Mitarbeiter sollen allen Quellkode 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 konsisten Stil zu schreiben.
- Alle Mitarbeiter sollen allen Quellkode 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 konsisten Stil zu schreiben.
- Alle Mitarbeiter sollen allen Quellkode 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*⁵⁹, die wir unter <https://peps.python.org/pep-0008> finden können. Python-Kode der PEP8 verletzt ist falscher Python-Kode.



LIU Hui's Methode, π zu Approximieren



Die Irrationale Kreiszahl π



- Probieren wir jetzt ein ernsthafteres Beispiel.

Die Irrationale Kreiszahl π



- Probieren wir jetzt ein ernsthafteres Beispiel.
- Ich bin nicht besonders gut in Mathe, aber ich mag Mathe trotzdem sehr. Machen wir also ein Mathe-Beispiel.

Die Irrationale Kreiszahl π



- Probieren wir jetzt ein ernsthafteres Beispiel.
- Ich bin nicht besonders gut in Mathe, aber ich mag Mathe trotzdem sehr. Machen wir also ein Mathe-Beispiel.
- Die Kreiszahl π ist das Verhältnis vom Umfang des Kreises zu seinem Durchmesser.

Die Irrationale Kreiszahl π



- Probieren wir jetzt ein ernsthafteres Beispiel.
- Ich bin nicht besonders gut in Mathe, aber ich mag Mathe trotzdem sehr. Machen wir also ein Mathe-Beispiel.
- Die Kreiszahl π ist das Verhältnis vom Umfang des Kreises zu seinem Durchmesser.
- Wir haben bereits in Einheit 8 über den Datentyp `float` gesagt, dass π transzendent ist, eine unendliche, sich nicht wiederholende Sequenz von Ziffern.

Die Irrationale Kreiszahl π



- Probieren wir jetzt ein ernsthafteres Beispiel.
- Ich bin nicht besonders gut in Mathe, aber ich mag Mathe trotzdem sehr. Machen wir also ein Mathe-Beispiel.
- Die Kreiszahl π ist das Verhältnis vom Umfang des Kreises zu seinem Durchmesser.
- Wir haben bereits in Einheit 8 über den Datentyp `float` gesagt, dass π transzendent ist, eine unendliche, sich nicht wiederholende Sequenz von Ziffern.
- Wir können π bis zu einer gewissen Präzision berechnen, z. B. als die `float`-Konstante `pi` mit dem Wert `3.141592653589793`.

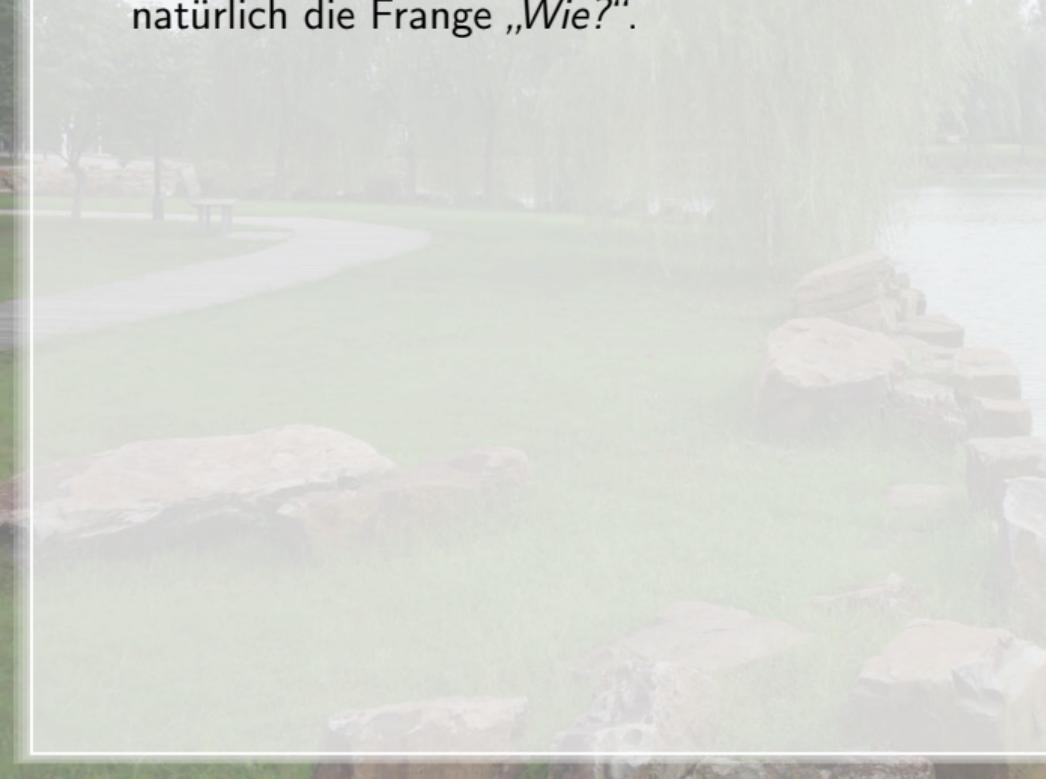
Die Irrationale Kreiszahl π



- Probieren wir jetzt ein ernsthafteres Beispiel.
- Ich bin nicht besonders gut in Mathe, aber ich mag Mathe trotzdem sehr. Machen wir also ein Mathe-Beispiel.
- Die Kreiszahl π ist das Verhältnis vom Umfang des Kreises zu seinem Durchmesser.
- Wir haben bereits in Einheit 8 über den Datentyp `float` gesagt, dass π transzendent ist, eine unendliche, sich nicht wiederholende Sequenz von Ziffern.
- Wir können π bis zu einer gewissen Präzision berechnen, z. B. als die `float`-Konstante `pi` mit dem Wert `3.141592653589793`.
- Aber wir können π niemals vollständig hinschreiben.

LIU Hui's Methode, π zu Approximieren

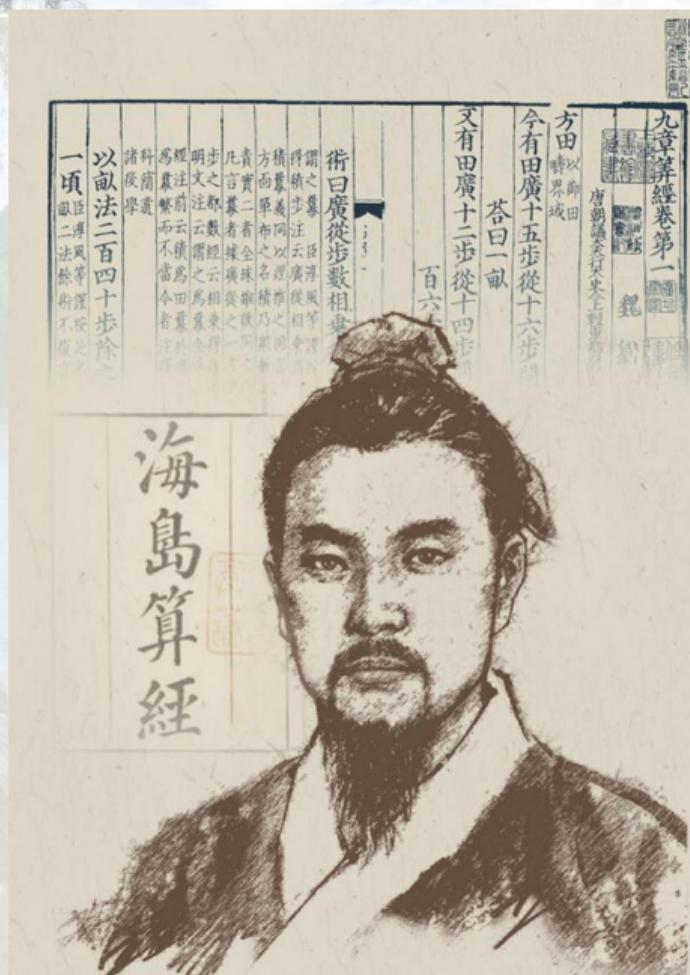
- Wenn ich sage „Wir können π bis zu einer gewissen Präzision berechnen.“ dann stellt sich natürlich die Frage „Wie?“.



Moderne Darstellung von LIU Hui (劉徽) aus [67]. Nicht unter der Creative Commons Lizenz, Copyright © liegt bei CAST / dem offiziellen WeChat-Account von VOC.

LIU Hui's Methode, π zu Approximieren

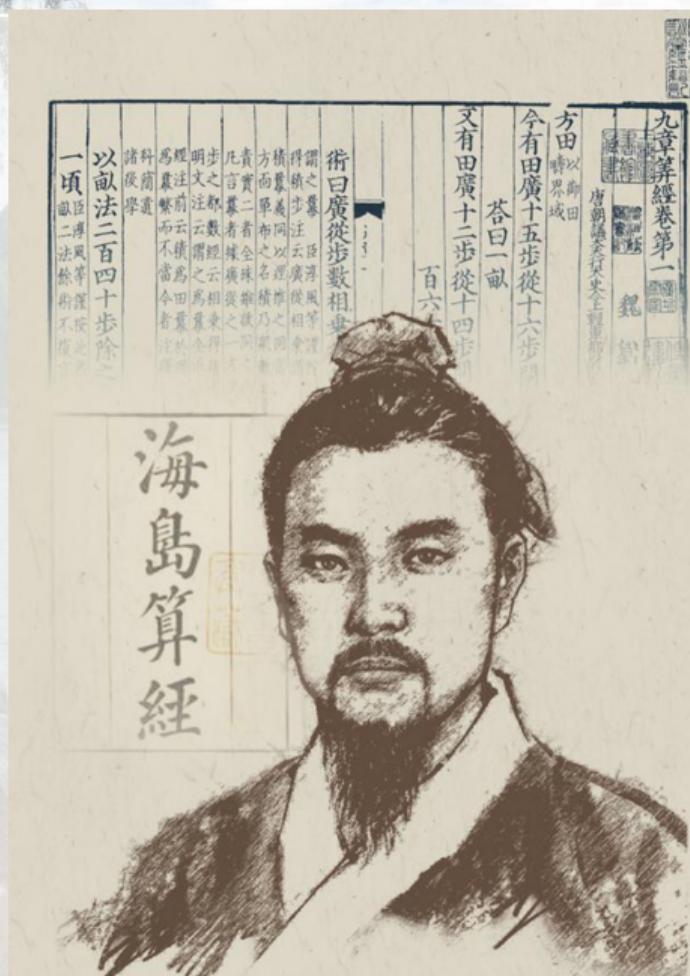
- Wenn ich sage „Wir können π bis zu einer gewissen Präzision berechnen.“ dann stellt sich natürlich die Frage „Wie?“.
- Eine besonders geniale Antwort ist uns von dem chinesischen Mathematiker LIU Hui (刘徽) irgendwann im dritten Jahrhundert Common Era (CE)^{37,67} in seinen Kommentaren zu dem berühmten chinesischen Mathematikbuch *Jiu Zhang Suanshu* (九章算术)^{10,12,29,37,52} gegeben worden.



Moderne Darstellung von LIU Hui (刘徽) aus [67]. Nicht unter der Creative Commons Lizenz, Copyright © liegt bei CAST / dem offiziellen WeChat-Account von VOC.

LIU Hui's Methode, π zu Approximieren

- Wenn ich sage „Wir können π bis zu einer gewissen Präzision berechnen.“ dann stellt sich natürlich die Frage „Wie?“.
- Eine besonders geniale Antwort ist uns von dem chinesischen Mathematiker LIU Hui (刘徽) irgendwann im dritten Jahrhundert Common Era (CE)^{37,67} in seinen Kommentaren zu dem berühmten chinesischen Mathematikbuch *Jiu Zhang Suanshu* (九章算术)^{10,12,29,37,52} gegeben worden.
- Die Idee ist, reguläre e -Ecke mit steigener Anzahl e der Ecken in einen Kreis einzuschreiben, so dass die Ecken jeweils auf dem Kreis liegen.



Moderne Darstellung von LIU Hui (刘徽) aus [67]. Nicht unter der Creative Commons Lizenz, Copyright © liegt bei CAST / dem offiziellen WeChat-Account von VOC.

LIU Hui's Methode, π zu Approximieren



- Eine besonders geniale Antwort ist uns von dem chinesischen Mathematiker LIU Hui (刘徽) irgendwann im dritten Jahrhundert Common Era (CE)^{37,67} in seinen Kommentaren zu dem berühmten chinesischen Mathematikbuch *Jiu Zhang Suanshu* (九章算术)^{10,12,29,37,52} gegeben worden.
- Die Idee ist, reguläre e -Ecke mit steigener Anzahl e der Ecken in einen Kreis einzuschreiben, so dass die Ecken jeweils auf dem Kreis liegen.
- Wir beginnen mit einem $e = 6$ -Eck.

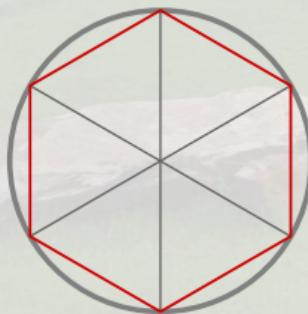


$e=6$

LIU Hui's Methode, π zu Approximieren



- Eine besonders geniale Antwort ist uns von dem chinesischen Mathematiker LIU Hui (刘徽) irgendwann im dritten Jahrhundert Common Era (CE)^{37,67} in seinen Kommentaren zu dem berühmten chinesischen Mathematikbuch *Jiu Zhang Suanshu* (九章算术)^{10,12,29,37,52} gegeben worden.
- Die Idee ist, reguläre e -Ecke mit steigener Anzahl e der Ecken in einen Kreis einzuschreiben, so dass die Ecken jeweils auf dem Kreis liegen.
- Wir beginnen mit einem $e = 6$ -Eck.
- Weil es ein reguläres Sechseck ist, können wir es in sechs Dreiecke teilen.

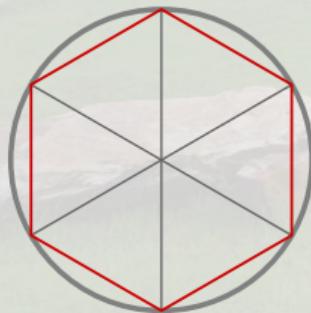


$e=6$

LIU Hui's Methode, π zu Approximieren



- Die Idee ist, reguläre e -Ecke mit steigener Anzahl e der Ecken in einen Kreis einzuschreiben, so dass die Ecken jeweils auf dem Kreis liegen.
- Wir beginnen mit einem $e = 6$ -Eck.
- Weil es ein reguläres Sechseck ist, können wir es in sechs Dreiecke teilen.
- Es sind gleichschenklige Dreiecke, weil zwei Seiten (mit der Länge r) im Mittelpunkt des Kreises beginnen und auf dem Kreis enden.

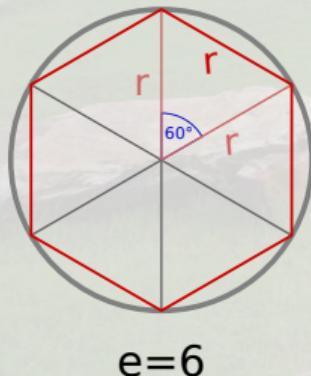


$e=6$

LIU Hui's Methode, π zu Approximieren



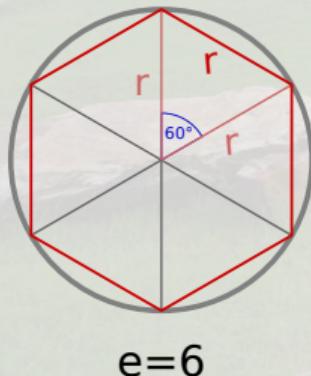
- Wir beginnen mit einem $e = 6$ -Eck.
- Weil es ein reguläres Sechseck ist, können wir es in sechs Dreiecke teilen.
- Es sind gleichschenklige Dreiecke, weil zwei Seiten (mit der Länge r) im Mittelpunkt des Kreises beginnen und auf dem Kreis enden.
- Der Winkel zwischen diesen Seiten beträgt 60° weil wir ja den Vollkreis auf sechs Dreiecke aufteilen ($360^\circ/6 = 60^\circ$).



LIU Hui's Methode, π zu Approximieren



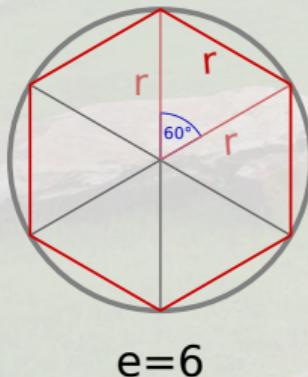
- Weil es ein reguläres Sechseck ist, können wir es in sechs Dreiecke teilen.
- Es sind gleichschenklige Dreiecke, weil zwei Seite (mit der Länge r) im Mittelpunkt des Kreises beginnen und auf dem Kreis enden.
- Der Winkel zwischen diesen Seiten beträgt 60° weil wir ja den Vollkreis auf sechs Dreiecke aufteilen ($360^\circ/6 = 60^\circ$).
- Deshalb sind die Dreiecke sogar gleichseitig und die dritte Seite hat auch Länge r .



LIU Hui's Methode, π zu Approximieren



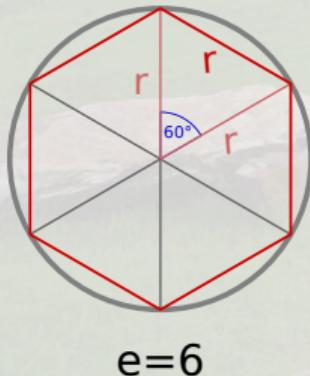
- Weil es ein reguläres Sechseck ist, können wir es in sechs Dreiecke teilen.
- Es sind gleichschenklige Dreiecke, weil zwei Seite (mit der Länge r) im Mittelpunkt des Kreises beginnen und auf dem Kreis enden.
- Der Winkel zwischen diesen Seiten beträgt 60° weil wir ja den Vollkreis auf sechs Dreiecke aufteilen ($360^\circ/6 = 60^\circ$).
- Deshalb sind die Dreiecke sogar gleichseitig und die dritte Seite hat auch Länge r .
- Damit haben also alle e Seiten s_6 des Sechseckes die Länge r .



LIU Hui's Methode, π zu Approximieren



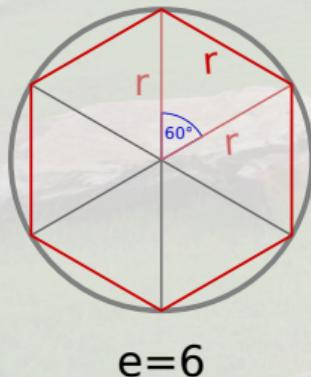
- Es sind gleichschenklige Dreiecke, weil zwei Seite (mit der Länge r) im Mittelpunkt des Kreises beginnen und auf dem Kreis enden.
- Der Winkel zwischen diesen Seiten beträgt 60° weil wir ja den Vollkreis auf sechs Dreiecke aufteilen ($360^\circ/6 = 60^\circ$).
- Deshalb sind die Dreiecke sogar gleichseitig und die dritte Seite hat auch Länge r .
- Damit haben also alle e Seiten s_6 des Sechseckes die Länge r .
- Der Umfang des Sechsecks ist also $U = e * s_6 = 6 * r$.



LIU Hui's Methode, π zu Approximieren



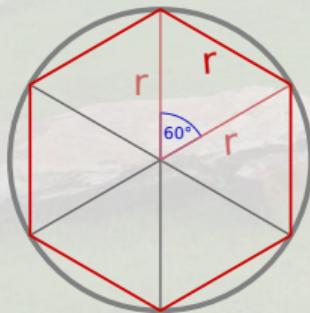
- Der Winkel zwischen diesen Seiten beträgt 60° weil wir ja den Vollkreis auf sechs Dreiecke aufteilen ($360^\circ/6 = 60^\circ$).
- Deshalb sind die Dreiecke sogar gleichseitig und die dritte Seite hat auch Länge r .
- Damit haben also alle e Seiten s_6 des Sechsecks die Länge r .
- Der Umfang des Sechsecks ist also $U = e * s_6 = 6 * r$.
- Der Durchmesser des Kreises ist $D = 2r$.



LIU Hui's Methode, π zu Approximieren



- Deshalb sind die Dreiecke sogar gleichseitig und die dritte Seite hat auch Länge r .
- Damit haben also alle e Seiten s_6 des Sechseckes die Länge r .
- Der Umfang des Sechsecks ist also $U = e * s_6 = 6 * r$.
- Der Durchmesser des Kreises ist $D = 2r$.
- Wenn wir den Umfang des Kreises durch den Umfang des Sechsecks annähern, könnten wir π annähern als $\pi \approx \frac{U}{D}$.

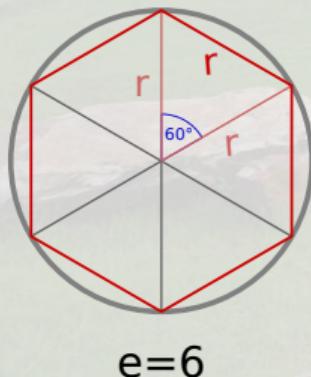


$$e=6$$

LIU Hui's Methode, π zu Approximieren



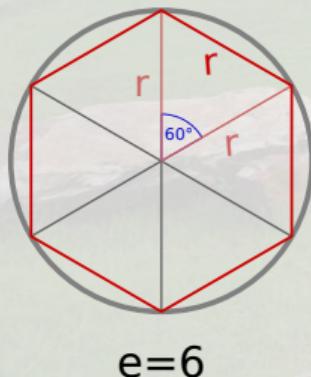
- Damit haben also alle e Seiten s_6 des Sechseckes die Länge r .
- Der Umfang des Sechsecks ist also $U = e * s_6 = 6 * r$.
- Der Durchmesser des Kreises ist $D = 2r$.
- Wenn wir den Umfang des Kreises durch den Umfang des Sechsecks annähern, könnten wir π annähern als $\pi \approx \frac{U}{D}$.
- Für $e = 6$ Ecken ergibt das $\pi_6 = \frac{6r}{2r} = 3$.



LIU Hui's Methode, π zu Approximieren



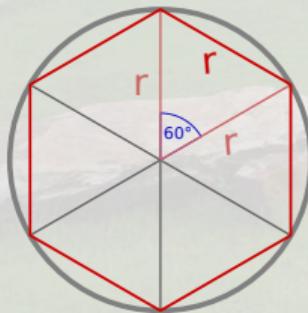
- Der Umfang des Sechsecks ist also $U = e * s_6 = 6 * r$.
- Der Durchmesser des Kreises ist $D = 2r$.
- Wenn wir den Umfang des Kreises durch den Umfang des Sechsecks annähern, könnten wir π annähern als $\pi \approx \frac{U}{D}$.
- Für $e = 6$ Ecken ergibt das $\pi_6 = \frac{6r}{2r} = 3$.
- Das ist eine eher ... grobe Annäherung von π .



LIU Hui's Methode, π zu Approximieren



- Der Umfang des Sechsecks ist also $U = e * s_6 = 6 * r$.
- Der Durchmesser des Kreises ist $D = 2r$.
- Wenn wir den Umfang des Kreises durch den Umfang des Sechsecks annähern, könnten wir π annähern als $\pi \approx \frac{U}{D}$.
- Für $e = 6$ Ecken ergibt das $\pi_6 = \frac{6r}{2r} = 3$.
- Das ist eine eher ... grobe Annäherung von π .
- Wir können näher rankommen, wenn wir mehr Ecken nehmen, also größere e .

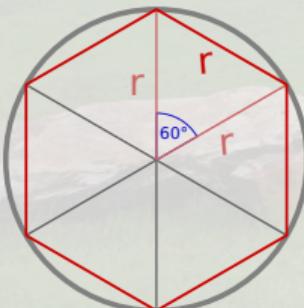


$e=6$

LIU Hui's Methode, π zu Approximieren



- Der Durchmesser des Kreises ist $D = 2r$.
- Wenn wir den Umfang des Kreises durch den Umfang des Sechsecks annähern, könnten wir π annähern als $\pi \approx \frac{U}{D}$.
- Für $e = 6$ Ecken ergibt das $\pi_6 = \frac{6r}{2r} = 3$.
- Das ist eine eher ... grobe Annäherung von π .
- Wir können näher rankommen, wenn wir mehr Ecken nehmen, also größere e .
- Die geniale Idee von LIU Hui (刘徽) war es, e -Ecke mit $e = 3 * 2^n$ Ecken zu benutzen.

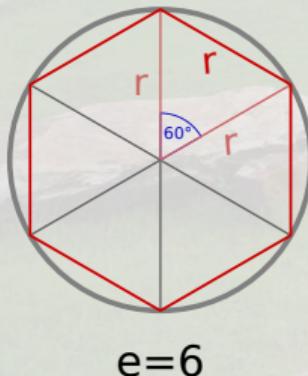


$e=6$

LIU Hui's Methode, π zu Approximieren

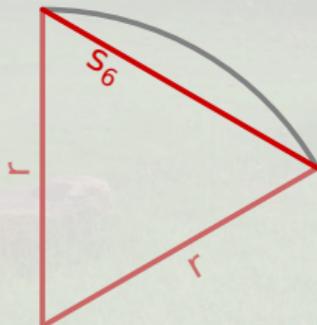
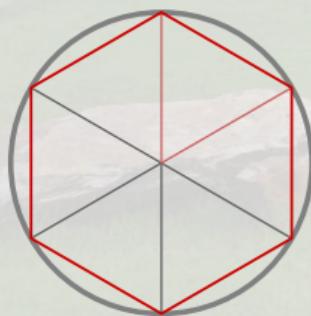


- Wenn wir den Umfang des Kreises durch den Umfang des Sechsecks annähern, könnten wir π annähern als $\pi \approx \frac{U}{D}$.
- Für $e = 6$ Ecken ergibt das $\pi_6 = \frac{6r}{2r} = 3$.
- Das ist eine eher ... grobe Annäherung von π .
- Wir können näher rankommen, wenn wir mehr Ecken nehmen, also größere e .
- Die geniale Idee von LIU Hui (刘徽) war es, e -Ecke mit $e = 3 * 2^n$ Ecken zu benutzen.
- Für $n = 1$, bekommen wir ein Sechseck mit $e = 6$.



LIU Hui's Methode, π zu Approximieren

- Für $e = 6$ Ecken ergibt das $\pi_6 = \frac{6r}{2r} = 3$.
- Das ist eine eher ... grobe Annäherung von π .
- Wir können näher rankommen, wenn wir mehr Ecken nehmen, also größere e .
- Die geniale Idee von LIU Hui (刘徽) war es, e -Ecke mit $e = 3 * 2^n$ Ecken zu benutzen.
- Für $n = 1$, bekommen wir ein Sechseck mit $e = 6$.
- Für $n = 2$ verdoppeln wir die Ecken und bekommen ein $e = 12$ -Eck.

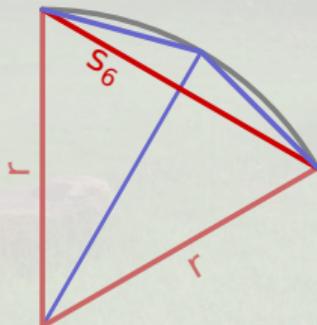
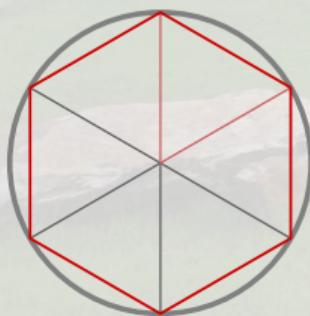


$e=6$



LIU Hui's Methode, π zu Approximieren

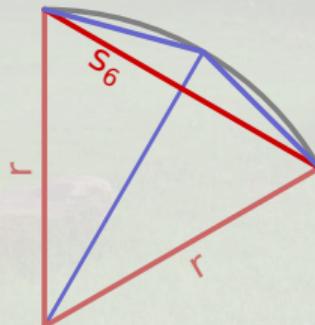
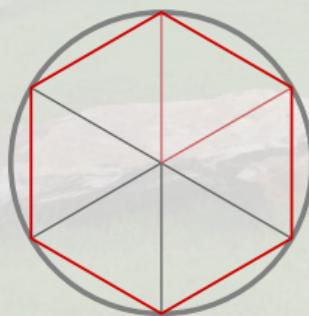
- Das ist eine eher ... grobe Annäherung von π .
- Wir können näher rankommen, wenn wir mehr Ecken nehmen, also größere e .
- Die geniale Idee von LIU Hui (刘徽) war es, e -Ecke mit $e = 3 * 2^n$ Ecken zu benutzen.
- Für $n = 1$, bekommen wir ein Sechseck mit $e = 6$.
- Für $n = 2$ verdoppeln wir die Ecken und bekommen ein $e = 12$ -Eck.
- Aber wie bekommen wir die Länge der Seiten s_{12} heraus?



LIU Hui's Methode, π zu Approximieren



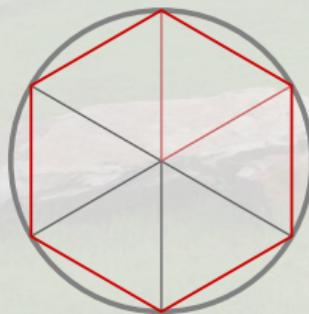
- Wir können näher rankommen, wenn wir mehr Ecken nehmen, also größere e .
- Die geniale Idee von LIU Hui (刘徽) war es, e -Ecke mit $e = 3 * 2^n$ Ecken zu benutzen.
- Für $n = 1$, bekommen wir ein Sechseck mit $e = 6$.
- Für $n = 2$ verdoppeln wir die Ecken und bekommen ein $e = 12$ -Eck.
- Aber wie bekommen wir die Länge der Seiten s_{12} heraus?
- Wir können sie von den Längen s_6 und dem Radius r berechnen.



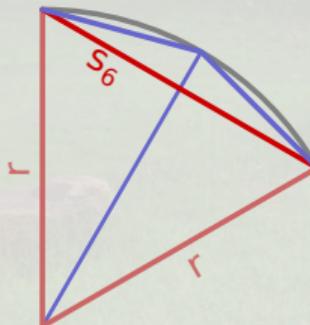


LIU Hui's Methode, π zu Approximieren

- Die geniale Idee von LIU Hui (刘徽) war es, e -Ecke mit $e = 3 * 2^n$ Ecken zu benutzen.
- Für $n = 1$, bekommen wir ein Sechseck mit $e = 6$.
- Für $n = 2$ verdoppeln wir die Ecken und bekommen ein $e = 12$ -Eck.
- Aber wie bekommen wir die Länge der Seiten s_{12} heraus?
- Wir können sie von den Längen s_6 und dem Radius r berechnen.
- Wir verwenden die selben 6 Ecken des Sechsecks und fügen nochmal 6 Ecken hinzu.



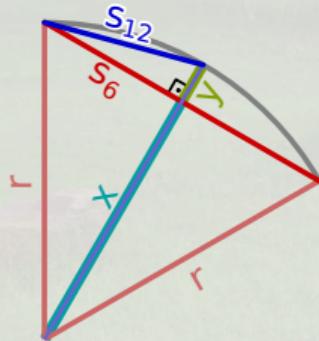
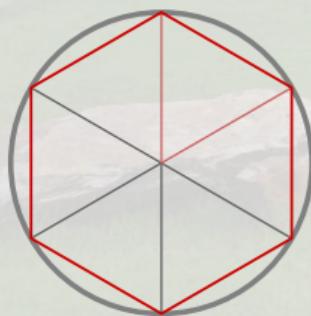
$e=6$



LIU Hui's Methode, π zu Approximieren

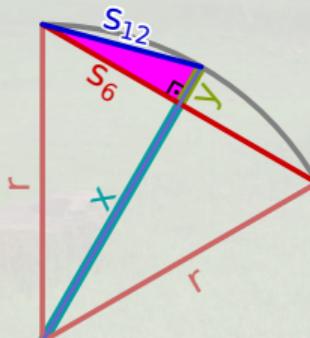
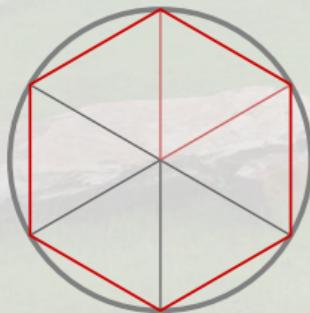


- Für $n = 2$ verdoppeln wir die Ecken und bekommen ein $e = 12$ -Eck.
- Aber wie bekommen wir die Länge der Seiten s_{12} heraus?
- Wir können sie von den Längen s_6 und dem Radius r berechnen.
- Wir verwenden die selben 6 Ecken des Sechsecks und fügen nochmal 6 Ecken hinzu.
- Wenn wir diese Ecken mit dem Zentrum des Kreises verbinden, dann zerschneiden die neuen Linien die Seiten des Sechsecks genau in zwei gleich große Hälften und tut dies in einem 90° Winkel.



LIU Hui's Methode, π zu Approximieren

- Wir können sie von den Längen s_6 und dem Radius r berechnen.
- Wir verwenden die selben 6 Ecken des Sechsecks und fügen nochmal 6 Ecken hinzu.
- Wenn wir diese Ecken mit dem Zentrum des Kreises verbinden, dann zerschneiden die neue Linie die Seiten des Sechsecks genau in zwei gleich große Hälften und tut dies in einem 90° Winkel.
- Die neue Seitenlänge s_{12} ist die Hypotenuse eines rechtwinkligen Dreiecks mit Basis $\frac{s_6}{2}$ und Höhe y .

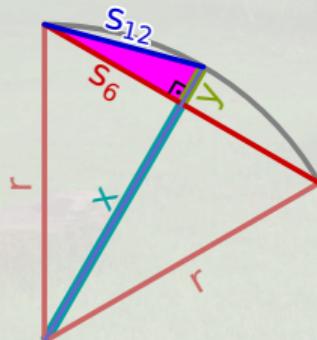
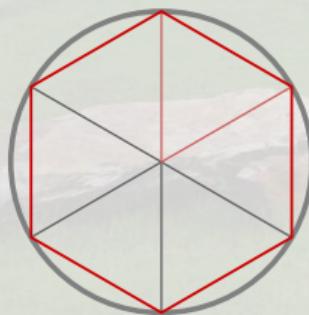


$$e=6$$



LIU Hui's Methode, π zu Approximieren

- Wenn wir diese Ecken mit dem Zentrum des Kreises verbinden, dann zerschneiden die neue Linie die Seiten des Sechsecks genau in zwei gleich große Hälften und tut dies in einem 90° Winkel.
- Die neue Seitenlänge s_{12} ist die Hypotenuse eines rechtwinkligen Dreiecks mit Basis $\frac{s_6}{2}$ und Höhe y .
- Wir wissen, dass $r = x + y$.

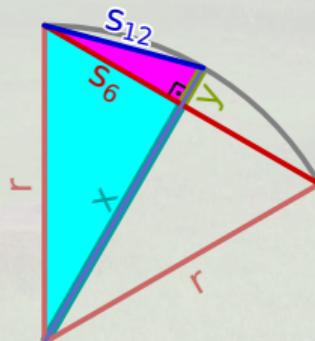
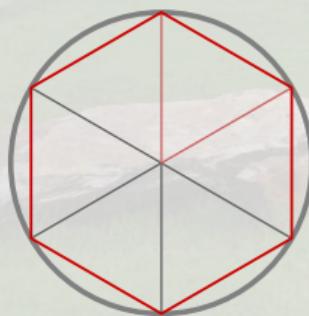


$$e=6$$

LIU Hui's Methode, π zu Approximieren



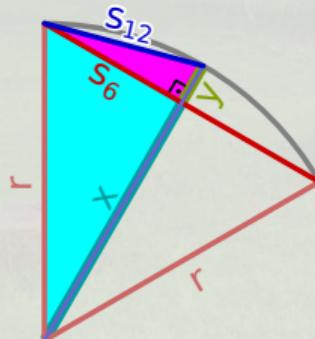
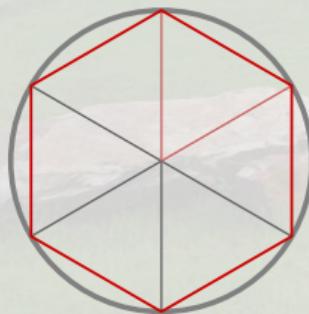
- Die neue Seitenlänge s_{12} ist die Hypotenuse eines rechtwinkligen Dreiecks mit Basis $\frac{s_6}{2}$ und Höhe y .
- Wir wissen, dass $r = x + y$.
- Es gibt auch noch ein zweites rechtwinkliges Dreieck, nämlich das mit Basis x , Höhe $\frac{s_6}{2}$, und Hypotenuse r .





LIU Hui's Methode, π zu Approximieren

- Die neue Seitenlänge s_{12} ist die Hypotenuse eines rechtwinkligen Dreiecks mit Basis $\frac{s_6}{2}$ und Höhe y .
- Wir wissen, dass $r = x + y$.
- Es gibt auch noch ein zweites rechtwinkliges Dreieck, nämlich das mit Basis x , Höhe $\frac{s_6}{2}$, und Hypotenuse r .
- Das gibt uns $x^2 + \left(\frac{s_6}{2}\right)^2 = r^2$.

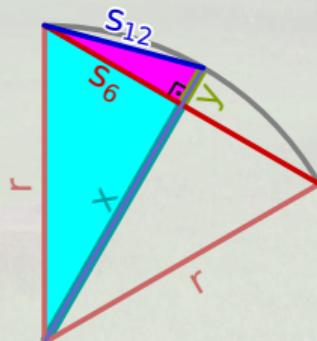
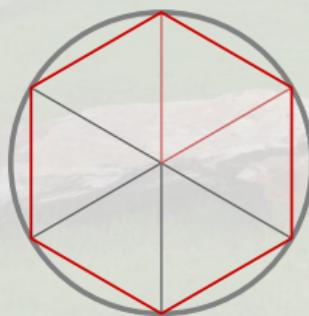


$$e=6$$

LIU Hui's Methode, π zu Approximieren



- Wir wissen, dass $r = x + y$.
- Es gibt auch noch ein zweites rechtwinkliges Dreieck, nämlich das mit Basis x , Höhe $\frac{s_6}{2}$, und Hypotenuse r .
- Das gibt uns $x^2 + \left(\frac{s_6}{2}\right)^2 = r^2$.
- Nehmen wir von jetzt an der Einfachheit halber $r = 1$.

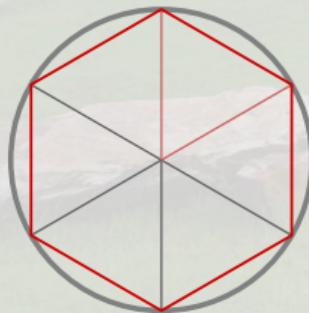


$$e=6$$

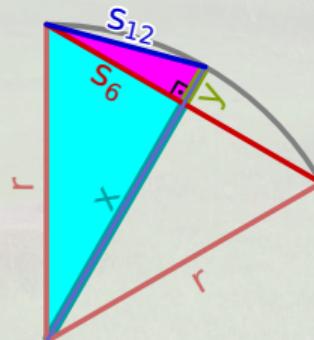
LIU Hui's Methode, π zu Approximieren



- Wir wissen, dass $r = x + y$.
- Es gibt auch noch ein zweites rechtwinkliges Dreieck, nämlich das mit Basis x , Höhe $\frac{s_6}{2}$, und Hypotenuse r .
- Das gibt uns $x^2 + \left(\frac{s_6}{2}\right)^2 = r^2$.
- Nehmen wir von jetzt an der Einfachheit halber $r = 1$.
- Wir bekommen also $x^2 = 1 - \left(\frac{s_6}{2}\right)^2 = 1 - \frac{s_6^2}{4}$ also $x = \sqrt{1 - \frac{s_6^2}{4}}$.

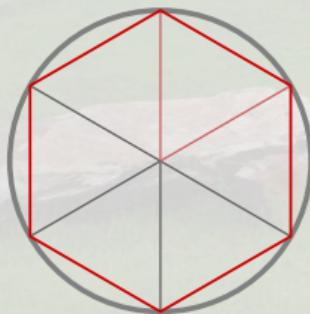


$$e=6$$

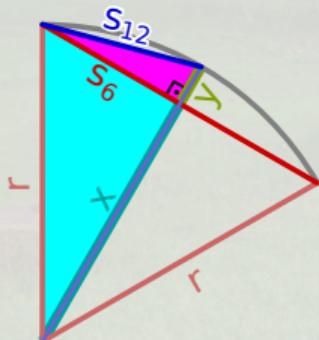


LIU Hui's Methode, π zu Approximieren

- Das gibt uns $x^2 + \left(\frac{s_6}{2}\right)^2 = r^2$.
- Nehmen wir von jetzt an der Einfachheit halber $r = 1$.
- Wir bekommen also $x^2 = 1 - \left(\frac{s_6}{2}\right)^2 = 1 - \frac{s_6^2}{4}$ also $x = \sqrt{1 - \frac{s_6^2}{4}}$.
- Da $y = r - x = 1 - x$ haben wir also $y = 1 - \sqrt{1 - \frac{s_6^2}{4}}$.

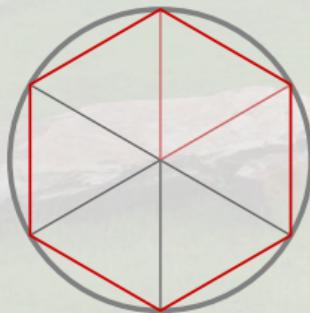


$$e=6$$

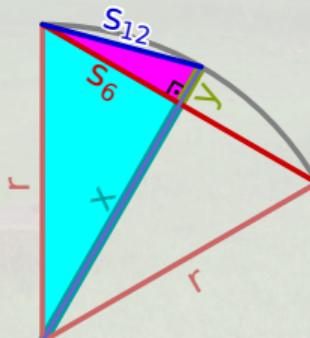


LIU Hui's Methode, π zu Approximieren

- Wir bekommen also $x^2 = 1 - \left(\frac{s_6}{2}\right)^2 = 1 - \frac{s_6^2}{4}$ also $x = \sqrt{1 - \frac{s_6^2}{4}}$.
- Da $y = r - x = 1 - x$ haben wir also $y = 1 - \sqrt{1 - \frac{s_6^2}{4}}$.
- Wir können jetzt also zu $s_{12}^2 = y^2 + \left(\frac{s_6}{2}\right)^2$ gehen, woraus nun $s_{12}^2 = \left(1 - \sqrt{1 - \frac{s_6^2}{4}}\right)^2 + \frac{s_6^2}{4}$ wird.

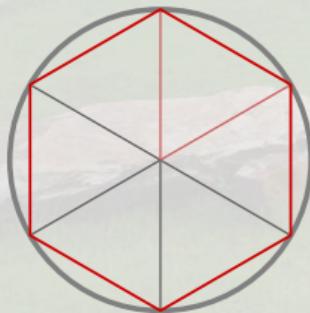


$$e=6$$

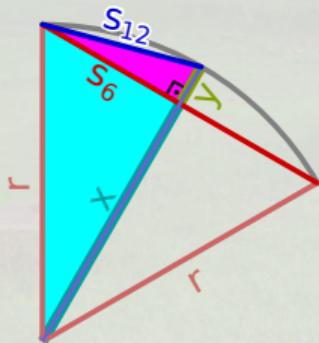


LIU Hui's Methode, π zu Approximieren

- Wir können jetzt also zu $s_{12}^2 = y^2 + \left(\frac{s_6}{2}\right)^2$ gehen, woraus nun $s_{12}^2 = \left(1 - \sqrt{1 - \frac{s_6^2}{4}}\right)^2 + \frac{s_6^2}{4}$ wird.
- Wir wenden $(a - b)^2 = a^2 - 2ab + b^2$ auf den ersten Term an und bekommen somit $s_{12}^2 = 1 - 2\sqrt{1 - \frac{s_6^2}{4}} + \left(1 - \frac{s_6^2}{4}\right) + \frac{s_6^2}{4}$.



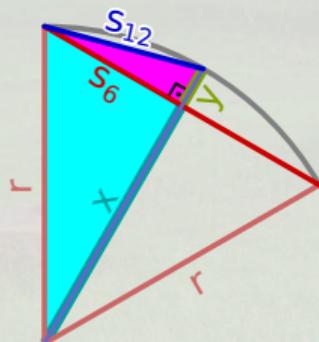
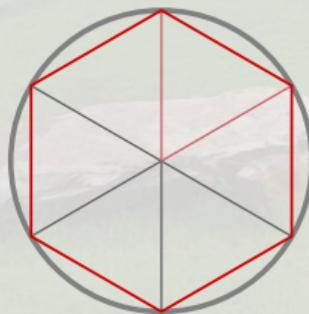
$$e=6$$



LIU Hui's Methode, π zu Approximieren

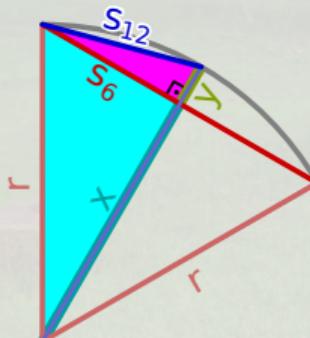
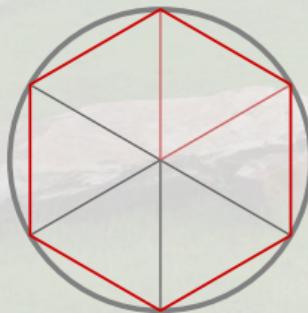


- Wir wenden $(a - b)^2 = a^2 - 2ab + b^2$ auf den ersten Term an und bekommen somit $s_{12}^2 = 1 - 2\sqrt{1 - \frac{s_6^2}{4}} + \left(1 - \frac{s_6^2}{4}\right) + \frac{s_6^2}{4}$.
- Damit bekommen wir $s_{12}^2 = 2 - 2\sqrt{1 - \frac{s_6^2}{4}} - \frac{s_6^2}{4} + \frac{s_6^2}{4}$, was wir dann weiter zu $s_{12}^2 = 2 - 2\sqrt{1 - \frac{s_6^2}{4}}$ verfeinern.



LIU Hui's Methode, π zu Approximieren

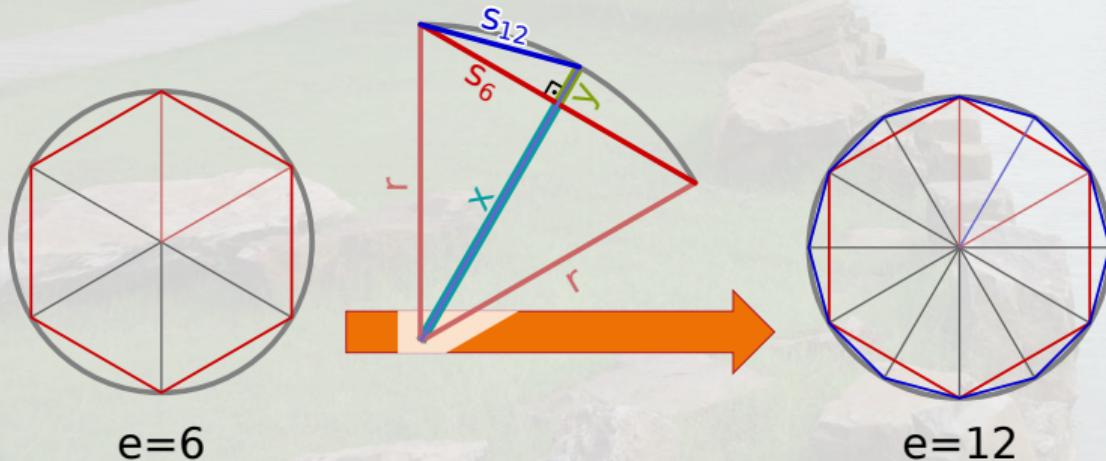
- Damit bekommen wir $s_{12}^2 = 2 - 2\sqrt{1 - \frac{s_6^2}{4}} - \frac{s_6^2}{4} + \frac{s_6^2}{4}$, was wir dann weiter zu $s_{12}^2 = 2 - 2\sqrt{1 - \frac{s_6^2}{4}}$ verfeinern.
- Wir ziehen nun die 2 von außerhalb der Wurzel in die Wurzel hinein, in dem wir alles mit $2^2 = 4$ multiplizieren und bekommen $s_{12}^2 = 2 - \sqrt{4 - s_6^2}$.



$$e=6$$

LIU Hui's Methode, π zu Approximieren

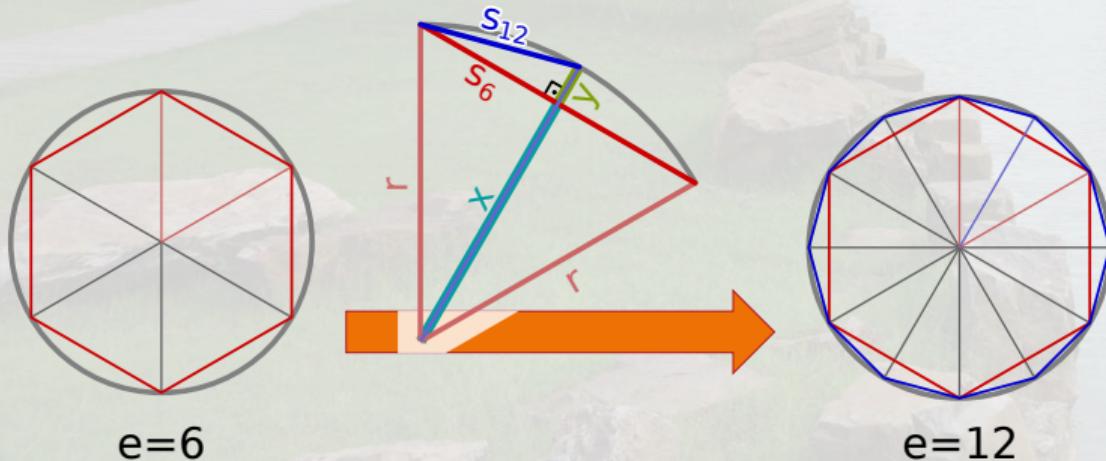
- Damit bekommen wir $s_{12}^2 = 2 - 2\sqrt{1 - \frac{s_6^2}{4}} - \frac{s_6^2}{4} + \frac{s_6^2}{4}$, was wir dann weiter zu $s_{12}^2 = 2 - 2\sqrt{1 - \frac{s_6^2}{4}}$ verfeinern.
- Wir ziehen nun die 2 von außerhalb der Wurzel in die Wurzel hinein, in dem wir alles mit $2^2 = 4$ multiplizieren und bekommen $s_{12}^2 = 2 - \sqrt{4 - s_6^2}$.
- Wir haben also nun die wunderschöne Formel $s_{12} = \sqrt{2 - \sqrt{4 - s_6^2}}$.



LIU Hui's Methode, π zu Approximieren

- Wir ziehen nun die 2 von außerhalb der Wurzel in die Wurzel hinein, in dem wir alles mit $2^2 = 4$ multiplizieren und bekommen $s_{12}^2 = 2 - \sqrt{4 - s_6^2}$.
- Wir haben also nun die wunderschöne Formel $s_{12} = \sqrt{2 - \sqrt{4 - s_6^2}}$.
- Als neue Annäherung π_{12} habe wir somit

$$\frac{12 * s_{12}}{2r} = 6 * s_{12} = 6\sqrt{2 - \sqrt{4 - s_6^2}} = 6\sqrt{2 - \sqrt{4 - 1}} = 6\sqrt{2 - \sqrt{3}} \approx 3.105828539.$$



LIU Hui's Methode, π zu Approximieren

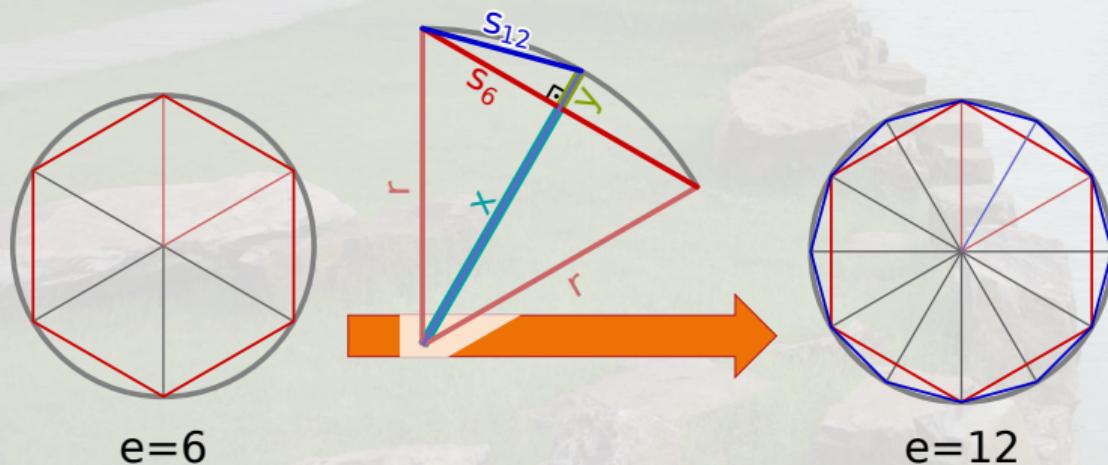


- Wir haben also nun die wunderschöne Formel $s_{12} = \sqrt{2 - \sqrt{4 - s_6^2}}$.

- Als neue Annäherung π_{12} habe wir somit

$$\frac{12 * s_{12}}{2r} = 6 * s_{12} = 6\sqrt{2 - \sqrt{4 - s_6^2}} = 6\sqrt{2 - \sqrt{4 - 1}} = 6\sqrt{2 - \sqrt{3}} \approx 3.105828539.$$

- Das ist schon ziemlich schön.



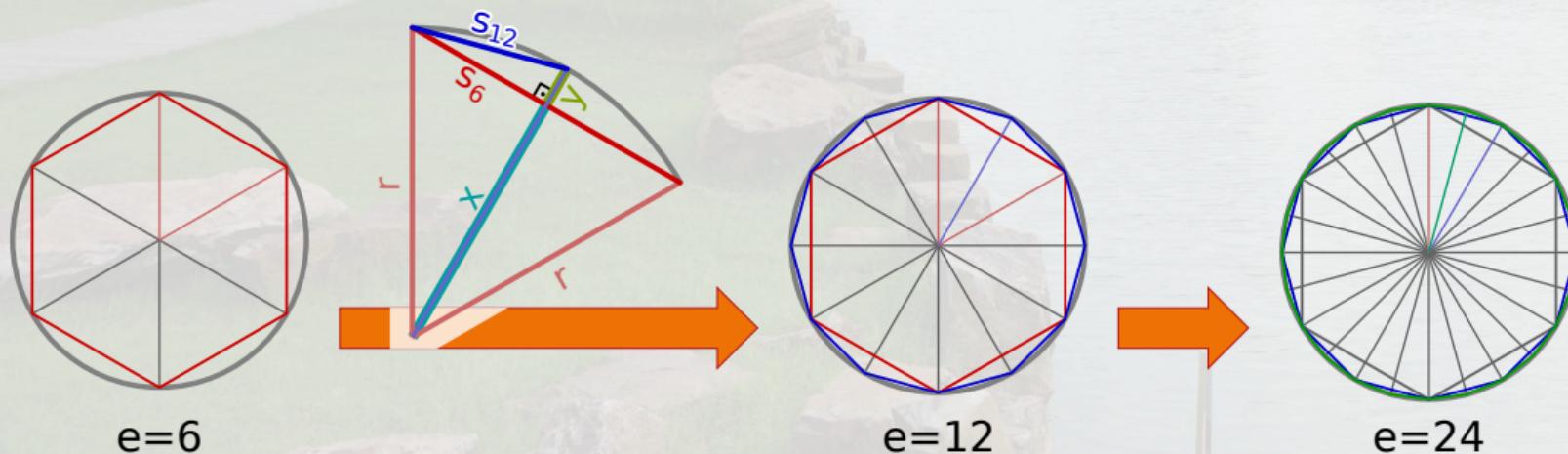
LIU Hui's Methode, π zu Approximieren



- Als neue Annäherung π_{12} habe wir somit

$$\frac{12 \cdot s_{12}}{2r} = 6 \cdot s_{12} = 6\sqrt{2 - \sqrt{4 - s_6^2}} = 6\sqrt{2 - \sqrt{4 - 1}} = 6\sqrt{2 - \sqrt{3}} \approx 3.105828539.$$

- Das ist schon ziemlich schön.
- Wir können diesen Schritt auch einfach wiederholen, um zu s_{24} zu kommen.



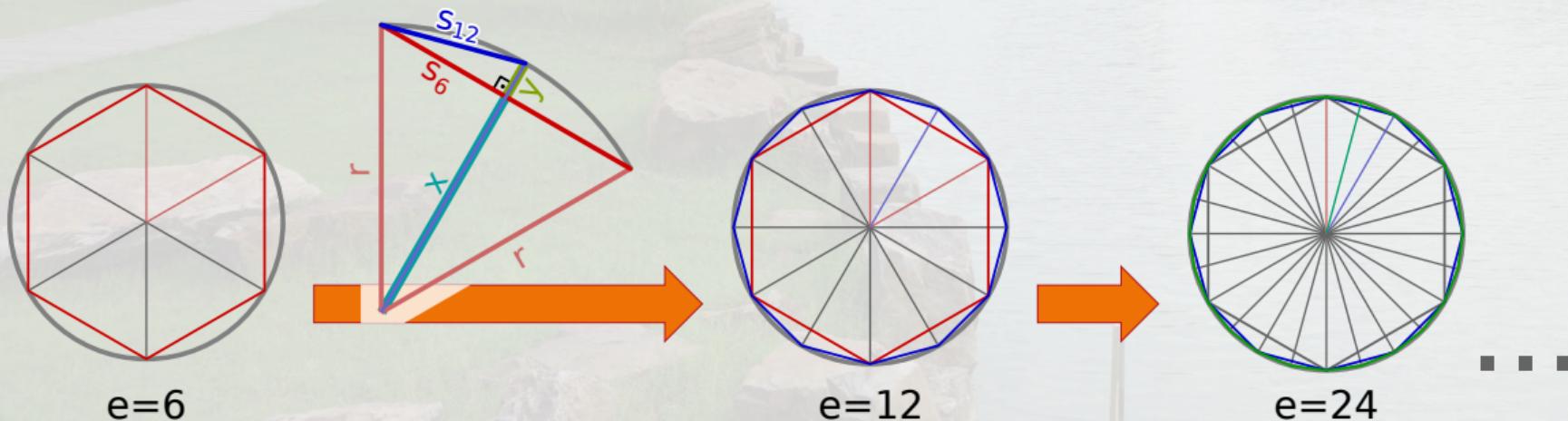
LIU Hui's Methode, π zu Approximieren



- Als neue Annäherung π_{12} habe wir somit

$$\frac{12 \cdot s_{12}}{2r} = 6 \cdot s_{12} = 6\sqrt{2 - \sqrt{4 - s_6^2}} = 6\sqrt{2 - \sqrt{4 - 1}} = 6\sqrt{2 - \sqrt{3}} \approx 3.105828539.$$

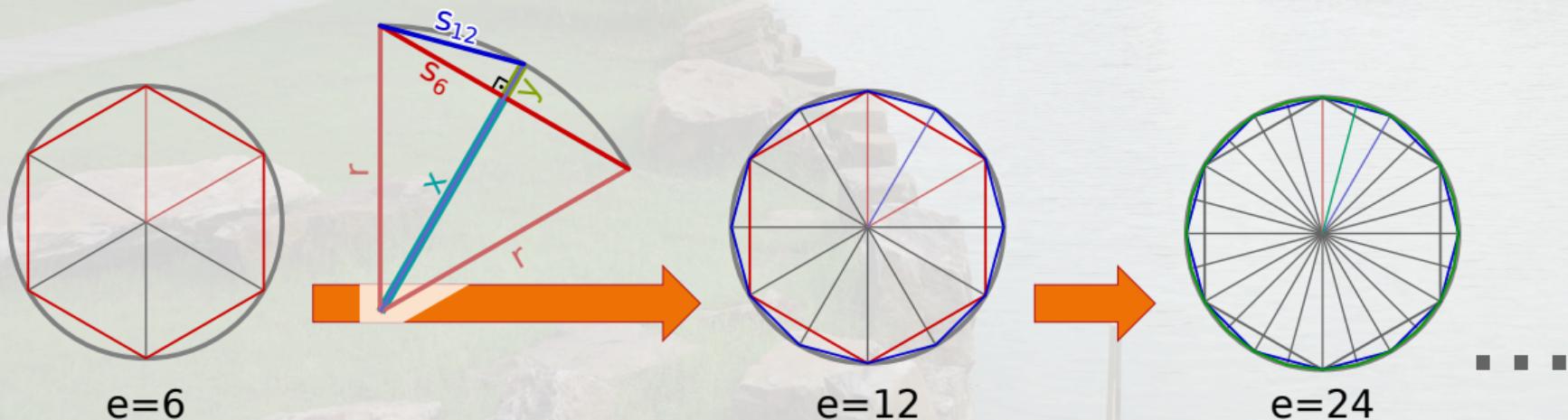
- Das ist schon ziemlich schön.
- Wir können diesen Schritt auch einfach wiederholen, um zu s_{24} zu kommen.
- Und dann können wir das wieder und wieder wiederholen.



LIU Hui's Methode, π zu Approximieren



- Das ist schon ziemlich schön.
- Wir können diesen Schritt auch einfach wiederholen, um zu s_{24} zu kommen.
- Und dann können wir das wieder und wieder wiederholen.
- Wir bekommen folgende Gleichungen.

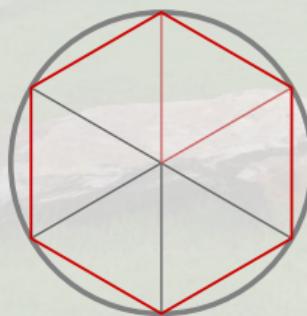


LIU Hui's Methode, π zu Approximieren

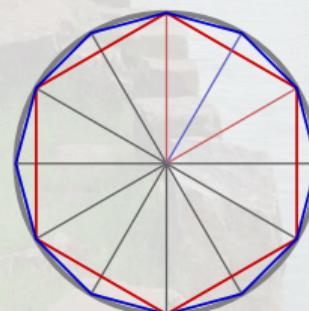
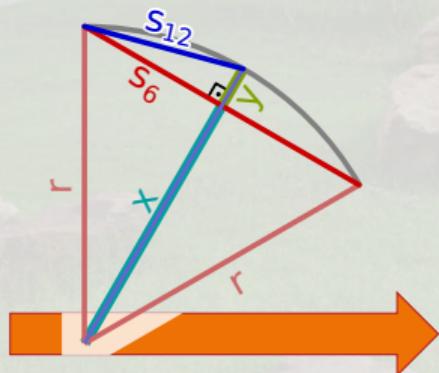
- Wir können diesen Schritt auch einfach wiederholen, um zu s_{24} zu kommen.
- Und dann können wir das wieder und wieder wiederholen.
- Wir bekommen folgende Gleichungen.

$$s_{2e} = \sqrt{2 - \sqrt{4 - s_e^2}}$$

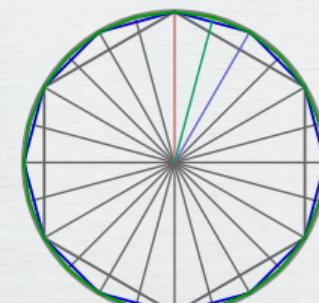
(2)



$e=6$



$e=12$



$e=24$

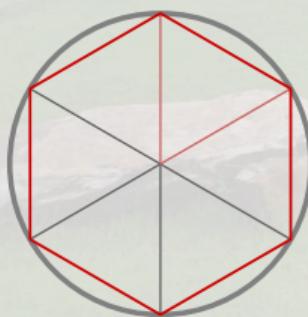
...

LIU Hui's Methode, π zu Approximieren

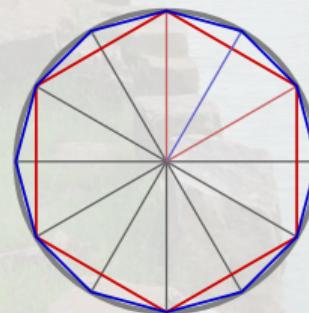
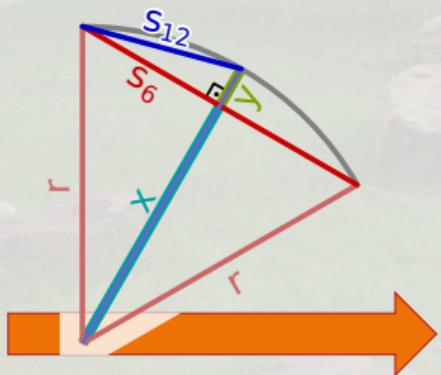
- Und dann können wir das wieder und wieder wiederholen.
- Wir bekommen folgende Gleichungen.

$$s_{2e} = \sqrt{2 - \sqrt{4 - s_e^2}} \quad (1)$$

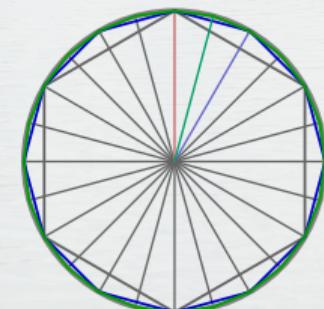
$$\pi_{2e} = \frac{e}{2} s_{2e} \quad (2)$$



$e=6$



$e=12$



$e=24$

...

LIU Hui's Methode in Python

- Jetzt wo wir schon etwas Programmieren gelernt haben, brauchen wir LIU Hui's Methode nicht in den Taschenrechner einzutippen.



LIU Hui's Methode in Python



- Jetzt wo wir schon etwas Programmieren gelernt haben, brauchen wir LIU Hui's Methode nicht in den Taschenrechner einzutippen.
- Wir werden auch Python nicht als Taschenrechner verwenden.

LIU Hui's Methode in Python



- Jetzt wo wir schon etwas Programmieren gelernt haben, brauchen wir LIU Hui's Methode nicht in den Taschenrechner einzutippen.
- Wir werden auch Python nicht als Taschenrechner verwenden.
- Stattdessen können wir alle notwendigen Berechnungen in eine Programmdatei eintippen.

LIU Hui's Methode in Python



- Jetzt wo wir schon etwas Programmieren gelernt haben, brauchen wir LIU Hui's Methode nicht in den Taschenrechner einzutippen.
- Wir werden auch Python nicht als Taschenrechner verwenden.
- Stattdessen können wir alle notwendigen Berechnungen in eine Programmdatei eintippen.
- We nennen diese `pi_liu_hui.py`.

LIU Hui's Methode in Python

```
1  from math import pi, sqrt    # We need sqrt. pi is for comparison.  
2  
3  # We use f-strings with Unicode escapes to print the current result.  
4  # "\u03c0" is the Unicode escape for the Greek letter pi.  
5  # "\u2248" is the Unicode escape for the "approximately equal" sign.  
6  print(f"We use Liu Hui's Method to Approximate \u03c0\u2248{pi}.")  
7  e = 6      # the number of edges: We start with a hexagon, i.e., e=6.  
8  s = 1.0    # the side length: Initially 1, meaning the radius is also 1.  
9  print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")  
10  
11 e *= 2    # We double the number of edges... ...now there are 12.  
12 s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.  
13 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")  
14  
15 e *= 2    # We double the number of edges... ...now there are 24.  
16 s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.  
17 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")  
18  
19 e *= 2    # We double the number of edges... ...now there are 48.  
20 s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.  
21 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")  
22  
23 e *= 2    # We double the number of edges... ...now there are 96.  
24 s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.  
25 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")  
26  
27 e *= 2    # We double the number of edges... ...now there are 192.  
28 s = sqrt(2 - sqrt(4 - (s ** 2)))  
29 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")
```



LIU Hui's Methode in Python



- Wir beginnen damit, die anfängliche Anzahl der Ecken auf $e = 6$ und die anfängliche Seitenlänge auf $s = 1$ zu setzen, denn wir wählen ja $r = 1$.
- In jedem Annäherungsschritt setzen wir dann $e *= 2$, was das selbe ist wie $e = e * 2$, um die Anzahl der Ecken zu verdoppeln.

```
from math import pi, sqrt    # We need sqrt. pi is for comparison.

# We use f-strings with Unicode escapes to print the current result.
# "\u03c0" is the Unicode escape for the Greek letter pi.
# "\u2248" is the Unicode escape for the "approximately equal" sign.
print(f"We use Liu Hui's Method to Approximate \u03c0\u2248{pi}.")
e = 6      # the number of edges: We start with a hexagon, i.e., e=6.
s = 1.0    # the side length: Initially 1, meaning the radius is also 1.
print(f"\u2113 edges, side length=\u2113 give \u03c0\u2248{e * s / 2}.")

e *= 2    # We double the number of edges... ...now there are 12.
s = sqrt(2 - sqrt(4 - (s ** 2)))  # ...and recompute the side length.
print(f"\u2113 edges, side length=\u2113 give \u03c0\u2248{e * s / 2}.)
```

LIU Hui's Methode in Python



- Wir beginnen damit, die anfängliche Anzahl der Ecken auf $e = 6$ und die anfängliche Seitenlänge auf $s = 1$ zu setzen, denn wir wählen ja $r = 1$.
- In jedem Annäherungsschritt setzen wir dann $e *= 2$, was das selbe ist wie $e = e * 2$, um die Anzahl der Ecken zu verdoppeln.
- Wir berechnen dann $s = \sqrt{2 - \sqrt{4 - (s ** 2)}}.$

```
from math import pi, sqrt    # We need sqrt. pi is for comparison.

# We use f-strings with Unicode escapes to print the current result.
# "\u03c0" is the Unicode escape for the Greek letter pi.
# "\u2248" is the Unicode escape for the "approximately equal" sign.
print(f"We use Liu Hui's Method to Approximate \u03c0\u2248{pi}.")
e = 6      # the number of edges: We start with a hexagon, i.e., e=6.
s = 1.0    # the side length: Initially 1, meaning the radius is also 1.
print(f"\u207ae edges, side length=\u207as give \u03c0\u2248{e * s / 2}.")

e *= 2    # We double the number of edges... ...now there are 12.
s = sqrt(2 - sqrt(4 - (s ** 2)))  # ...and recompute the side length.
print(f"\u207ae edges, side length=\u207as give \u03c0\u2248{e * s / 2}.)
```

LIU Hui's Methode in Python



- Wir beginnen damit, die anfängliche Anzahl der Ecken auf `e = 6` und die anfängliche Seitenlänge auf `s = 1` zu setzen, denn wir wählen ja $r = 1$.
- In jedem Annäherungsschritt setzen wir dann `e *= 2`, was das selbe ist wie `e = e * 2`, um die Anzahl der Ecken zu verdoppeln.
- Wir berechnen dann `s = sqrt(2 - sqrt(4 - (s ** 2)))`.
- Dafür müssen wir natürlich die `sqrt`-Funktion aus dem Modul `math` importieren.

```
from math import pi, sqrt    # We need sqrt. pi is for comparison.

# We use f-strings with Unicode escapes to print the current result.
# "\u03c0" is the Unicode escape for the Greek letter pi.
# "\u2248" is the Unicode escape for the "approximately equal" sign.
print(f"We use Liu Hui's Method to Approximate \u03c0\u2248{pi}.")
e = 6      # the number of edges: We start with a hexagon, i.e., e=6.
s = 1.0    # the side length: Initially 1, meaning the radius is also 1.
print(f"\u207ae edges, side length=\u207as give \u03c0\u2248{e * s / 2}.")

e *= 2    # We double the number of edges... ...now there are 12.
s = sqrt(2 - sqrt(4 - (s ** 2)))  # ...and recompute the side length.
print(f"\u207ae edges, side length=\u207as give \u03c0\u2248{e * s / 2}.")
```

LIU Hui's Methode in Python



- In jedem Annäherungsschritt setzen wir dann `e *= 2`, was das selbe ist wie `e = e * 2`, um die Anzahl der Ecken zu verdoppeln.
- Wir berechnen dann `s = sqrt(2 - sqrt(4 - (s ** 2)))`.
- Dafür müssen wir natürlich die `sqrt`-Funktion aus dem Modul `math` importieren.
- Wir geben dann den geschätzten Wert von π als `e * s / 2` aus.

```
from math import pi, sqrt    # We need sqrt. pi is for comparison.

# We use f-strings with Unicode escapes to print the current result.
# "\u03c0" is the Unicode escape for the Greek letter pi.
# "\u2248" is the Unicode escape for the "approximately equal" sign.
print(f"We use Liu Hui's Method to Approximate \u03c0\u2248{pi}.")
e = 6      # the number of edges: We start with a hexagon, i.e., e=6.
s = 1.0    # the side length: Initially 1, meaning the radius is also 1.
print(f"\u2113 edges, side length=\u2113{s} give \u03c0\u2248{e * s / 2}.")

e *= 2    # We double the number of edges... ...now there are 12.
s = sqrt(2 - sqrt(4 - (s ** 2)))  # ...and recompute the side length.
print(f"\u2113 edges, side length=\u2113{s} give \u03c0\u2248{e * s / 2}.)
```

LIU Hui's Methode in Python



- Wir berechnen dann `s = sqrt(2 - sqrt(4 - (s ** 2)))`.
- Dafür müssen wir natürlich die `sqrt`-Funktion aus dem Modul `math` importieren.
- Wir geben dann den geschätzten Wert von π als `e * s / 2` aus.
- Beachten Sie, wie elegant wir die Unicode-Zeichen π und \approx durch die Escape-Sequenzen `\u03c0` und `\u2248` darstellen.

```
from math import pi, sqrt    # We need sqrt. pi is for comparison.

# We use f-strings with Unicode escapes to print the current result.
# "\u03c0" is the Unicode escape for the Greek letter pi.
# "\u2248" is the Unicode escape for the "approximately equal" sign.
print(f"We use Liu Hui's Method to Approximate \u03c0\u2248{pi}.")
e = 6      # the number of edges: We start with a hexagon, i.e., e=6.
s = 1.0    # the side length: Initially 1, meaning the radius is also 1.
print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")

e *= 2    # We double the number of edges... ...now there are 12.
s = sqrt(2 - sqrt(4 - (s ** 2)))  # ...and recompute the side length.
print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")
```

LIU Hui's Methode in Python



- Wir geben dann den geschätzten Wert von π als `e * s / 2` aus.
- Beachten Sie, wie elegant wir die Unicode-Zeichen π und \approx durch die Escape-Sequenzen `\u03c0` und `\u2248` darstellen.
- So oder so, die Annäherungsschritte sind immer gleich, daher können wir den Kode einfach ein paar Mal kopieren.

```
from math import pi, sqrt    # We need sqrt. pi is for comparison.

# We use f-strings with Unicode escapes to print the current result.
# "\u03c0" is the Unicode escape for the Greek letter pi.
# "\u2248" is the Unicode escape for the "approximately equal" sign.
print(f"We use Liu Hui's Method to Approximate \u03c0\u2248{pi}.")
e = 6      # the number of edges: We start with a hexagon, i.e., e=6.
s = 1.0    # the side length: Initially 1, meaning the radius is also 1.
print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")

e *= 2    # We double the number of edges... ...now there are 12.
s = sqrt(2 - sqrt(4 - (s ** 2)))  # ...and recompute the side length.
print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")
```

LIU Hui's Methode in Python

```
1 from math import pi, sqrt    # We need sqrt. pi is for comparison.  
2  
3 # We use f-strings with Unicode escapes to print the current result.  
4 # "\u03c0" is the Unicode escape for the Greek letter pi.  
5 # "\u2248" is the Unicode escape for the "approximately equal" sign.  
6 print(f"We use Liu Hui's Method to Approximate \u03c0\u2248{pi}.")  
7 e = 6      # the number of edges: We start with a hexagon, i.e., e=6.  
8 s = 1.0    # the side length: Initially 1, meaning the radius is also 1.  
9 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")  
10  
11 e *= 2    # We double the number of edges... ...now there are 12.  
12 s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.  
13 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")  
14  
15 e *= 2    # We double the number of edges... ...now there are 24.  
16 s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.  
17 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")  
18  
19 e *= 2    # We double the number of edges... ...now there are 48.  
20 s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.  
21 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")  
22  
23 e *= 2    # We double the number of edges... ...now there are 96.  
24 s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.  
25 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")  
26  
27 e *= 2    # We double the number of edges... ...now there are 192.  
28 s = sqrt(2 - sqrt(4 - (s ** 2)))  
29 print(f"{e} edges, side length={s} give \u03c0\u2248{e * s / 2}.")
```



LIU Hui's Methode in Python



- Das ist der Text, die das Programm auf den stdout ausgibt.

```
1 We use Liu Hui's Method to Approximate π≈3.141592653589793.  
2 6 edges, side length=1.0 give π≈3.0.  
3 12 edges, side length=0.5176380902050416 give π≈3.1058285412302498.  
4 24 edges, side length=0.2610523844401031 give π≈3.132628613281237.  
5 48 edges, side length=0.13080625846028635 give π≈3.139350203046872.  
6 96 edges, side length=0.0654381656435527 give π≈3.14103195089053.  
7 192 edges, side length=0.03272346325297234 give π≈3.1414524722853443.
```

LIU Hui's Methode in Python



- Das ist der Text, die das Programm auf den stdout ausgibt.
- Für 192 Ecken bekommen wir die Annäherung 3.1414524722853443.

```
1 We use Liu Hui's Method to Approximate π≈3.141592653589793.  
2 6 edges, side length=1.0 give π≈3.0.  
3 12 edges, side length=0.5176380902050416 give π≈3.1058285412302498.  
4 24 edges, side length=0.2610523844401031 give π≈3.132628613281237.  
5 48 edges, side length=0.13080625846028635 give π≈3.139350203046872.  
6 96 edges, side length=0.0654381656435527 give π≈3.14103195089053.  
7 192 edges, side length=0.03272346325297234 give π≈3.1414524722853443.
```

LIU Hui's Methode in Python



- Das ist der Text, die das Programm auf den `stdout` ausgibt.
- Für 192 Ecken bekommen wir die Annäherung `3.1414524722853443`.
- Die Konstante `pi` aus dem Modul `math` hat den Wert `3.141592653589793`.

```
1 We use Liu Hui's Method to Approximate π≈3.141592653589793.  
2 6 edges, side length=1.0 give π≈3.0.  
3 12 edges, side length=0.5176380902050416 give π≈3.1058285412302498.  
4 24 edges, side length=0.2610523844401031 give π≈3.132628613281237.  
5 48 edges, side length=0.13080625846028635 give π≈3.139350203046872.  
6 96 edges, side length=0.0654381656435527 give π≈3.14103195089053.  
7 192 edges, side length=0.03272346325297234 give π≈3.1414524722853443.
```

LIU Hui's Methode in Python



- Das ist der Text, die das Programm auf den `stdout` ausgibt.
- Für 192 Ecken bekommen wir die Annäherung `3.1414524722853443`.
- Die Konstante `pi` aus dem Modul `math` hat den Wert `3.141592653589793`.
- Die ersten vier Ziffern sind also schon korrekt und die Abweichung beträgt nur 0.0045%!

```
1 We use Liu Hui's Method to Approximate π≈3.141592653589793.  
2 6 edges, side length=1.0 give π≈3.0.  
3 12 edges, side length=0.5176380902050416 give π≈3.1058285412302498.  
4 24 edges, side length=0.2610523844401031 give π≈3.132628613281237.  
5 48 edges, side length=0.13080625846028635 give π≈3.139350203046872.  
6 96 edges, side length=0.0654381656435527 give π≈3.14103195089053.  
7 192 edges, side length=0.03272346325297234 give π≈3.1414524722853443.
```

LIU Hui's Methode in Python



- Das ist der Text, die das Programm auf den stdout ausgibt.
- Für 192 Ecken bekommen wir die Annäherung 3.1414524722853443.
- Die Konstante `pi` aus dem Modul `math` hat den Wert 3.141592653589793.
- Die ersten vier Ziffern sind also schon korrekt und die Abweichung beträgt nur 0.0045%!
- Wenn nur der gute alte LIU Hui (刘徽) das sehen könnte.

```
tweise@weise-laptop: /tmp$ python3 pi_liu_hui.py
We use Liu Hui's Method to Approximate π=3.141592653589793.
6 edges, side length=1 give us π=3.0.
12 edges, side length=0.5176380902050416 give us π=3.1058285412302498.
24 edges, side length=0.2610523844401031 give us π=3.132628613281237.
48 edges, side length=0.13080625846028635 give us π=3.139350203046872.
96 edges, side length=0.0654381656435527 give us π=3.14103195089053.
192 edges, side length=0.03272346325297234 give us π=3.1414524722853443.
tweise@weise-laptop: /tmp$ 
```

LIU Hui's Methode in Python



- Wenn nur der gute alte LIU Hui (刘徽) das sehen könnte.
- Nun führen wir das Programm im Terminal aus.

```
tweise@weise-laptop: /tmp$ python3 pi_liu_hui.py
We use Liu Hui's Method to Approximate π=3.141592653589793.
6 edges, side length=1 give us π=3.0.
12 edges, side length=0.5176380902050416 give us π=3.1058285412302498.
24 edges, side length=0.2610523844401031 give us π=3.132628613281237.
48 edges, side length=0.13080625846028635 give us π=3.139350203046872.
96 edges, side length=0.0654381656435527 give us π=3.14103195089053.
192 edges, side length=0.03272346325297234 give us π=3.1414524722853443.
tweise@weise-laptop: /tmp$ 
```

LIU Hui's Methode in Python



- Nun führen wir das Programm im Terminal aus.
- Unter Ubuntu Linux können wir ein Terminal durch Druck auf **Ctrl**+**Alt**+**T** öffnen, unter Microsoft Windows durch Druck auf **Windows**+**R**, dann Schreiben von **cmd**, dann Druck auf **Enter**.

A screenshot of a terminal window titled "tweise@weise-laptop: /tmp". The window contains the following text output:

```
tweise@weise-laptop:/tmp$ python3 pi_liu_hui.py
We use Liu Hui's Method to Approximate π=3.141592653589793.
6 edges, side length=1 give us π=3.0.
12 edges, side length=0.5176380902050416 give us π=3.1058285412302498.
24 edges, side length=0.2610523844401031 give us π=3.132628613281237.
48 edges, side length=0.13080625846028635 give us π=3.139350203046872.
96 edges, side length=0.0654381656435527 give us π=3.14103195089053.
192 edges, side length=0.03272346325297234 give us π=3.1414524722853443.
tweise@weise-laptop:/tmp$
```

LIU Hui's Methode in Python



- Nun führen wir das Programm im Terminal aus.
- Unter Ubuntu Linux können wir ein Terminal durch Druck auf **Ctrl**+**Alt**+**T** öffnen, unter Microsoft Windows durch Druck auf **Windows**+**R**, dann Schreiben von **cmd**, dann Druck auf **Enter**.
- Auf beiden Betriebssystemen gehen wir mit Hilfe des Kommandos **cd** in den Ordner mit der Programmdatei **pi_liu_hui.py**.

A screenshot of a terminal window titled "tweise@weise-laptop: /tmp". The window contains the following text:

```
tweise@weise-laptop:/tmp$ python3 pi_liu_hui.py
We use Liu Hui's Method to Approximate π=3.141592653589793.
6 edges, side length=1 give us π=3.0.
12 edges, side length=0.5176380902050416 give us π=3.1058285412302498.
24 edges, side length=0.2610523844401031 give us π=3.132628613281237.
48 edges, side length=0.13080625846028635 give us π=3.139350203046872.
96 edges, side length=0.0654381656435527 give us π=3.14103195089053.
192 edges, side length=0.03272346325297234 give us π=3.1414524722853443.
tweise@weise-laptop:/tmp$
```

The terminal has a dark background and light-colored text. It includes standard Linux terminal icons at the top right.

LIU Hui's Methode in Python



- Unter Ubuntu Linux können wir ein Terminal durch Druck auf **Ctrl**+**Alt**+**T** öffnen, unter Microsoft Windows durch Druck auf **Windows**+**R**, dann Schreiben von **cmd**, dann Druck auf **Enter**.
- Auf beiden Betriebssystemen gehen wir mit Hilfe des Kommandos **cd** in den Ordner mit der Programmdatei **pi_liu_hui.py**.
- Dort können wir diese dann durch **python3 pi_liu_hui.py** ausführen.

A screenshot of a terminal window titled "tweise@weise-laptop: /tmp". The window shows the command "python3 pi_liu_hui.py" being run and its output. The output describes the use of Liu Hui's Method to approximate pi, showing results for 6, 12, 24, 48, 96, and 192 edges. The terminal has a dark theme with light-colored text and icons.

```
tweise@weise-laptop:~/tmp$ python3 pi_liu_hui.py
We use Liu Hui's Method to Approximate pi=3.141592653589793.
6 edges, side length=1 give us pi=3.0.
12 edges, side length=0.5176380902050416 give us pi=3.1058285412302498.
24 edges, side length=0.2610523844401031 give us pi=3.132628613281237.
48 edges, side length=0.13080625846028635 give us pi=3.139350203046872.
96 edges, side length=0.0654381656435527 give us pi=3.14103195089053.
192 edges, side length=0.03272346325297234 give us pi=3.1414524722853443.
tweise@weise-laptop:~/tmp$
```

LIU Hui's Methode in Python



- Auf beiden Betriebssystemen gehen wir mit Hilfe des Kommandos `cd` in den Ordner mit der Programmdatei `pi_liu_hui.py`.
- Dort können wir diese dann durch `python3 pi_liu_hui.py` ausführen.
- Die Ausgabe ist natürlich die selbe.

```
tweise@weise-laptop: /tmp$ python3 pi_liu_hui.py
We use Liu Hui's Method to Approximate π=3.141592653589793.
6 edges, side length=1 give us π=3.0.
12 edges, side length=0.5176380902050416 give us π=3.1058285412302498.
24 edges, side length=0.2610523844401031 give us π=3.132628613281237.
48 edges, side length=0.13080625846028635 give us π=3.139350203046872.
96 edges, side length=0.0654381656435527 give us π=3.14103195089053.
192 edges, side length=0.03272346325297234 give us π=3.1414524722853443.
tweise@weise-laptop: /tmp$ 
```

LIU Hui's Methode in Python



- Alternativ können wir die Programmdatei auch in PyCharm der IDE öffnen.

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "01_variables". Inside, there is a folder "01_variables" containing files "assignment.py" and "pi_liu_hui.py".
- Code Editor:** The file "pi_liu_hui.py" is open. The code implements Liu Hui's method to approximate pi using a hexagon inscribed in a circle.
- Code Content:**

```
from math import pi, sqrt
print(f"We use Liu Hui's Method to Approximate \u03c0={pi}.")
e = 6 # the number of edges: We start with a hexagon, i.e., e=6.
s = 1 # the side length: Initially 1, meaning the radius is also 1.
print(f"\{e} edges, side length=\{s} give us \u03c0=\{e * s / 2}.")

e *= 2 # We double the number of edges...
s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.
print(f"\{e} edges, side length=\{s} give us \u03c0=\{e * s / 2}.")

e *= 2 # We double the number of edges.
s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.
print(f"\{e} edges, side length=\{s} give us \u03c0=\{e * s / 2}.")

e *= 2 # We double the number of edges.
s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.
print(f"\{e} edges, side length=\{s} give us \u03c0=\{e * s / 2}.")

e *= 2 # We double the number of edges.
s = sqrt(2 - sqrt(4 - (s ** 2)))
print(f"\{e} edges, side length=\{s} give us \u03c0=\{e * s / 2}.)
```
- Status Bar:** Shows the file path "01_variables > pi_liu_hui.py", the current time "19:1", encoding "LF", character set "UTF-8", indentation "4 spaces", Python version "Python 3.10", and a warning icon.

LIU Hui's Methode in Python



- Um sie auszuführen, können wir auf den Dateinamen im Project-View rechts-klicken und dann auf **Run 'pi_liu_hui'**.

LIU Hui's Methode in Python



- Oder wir drücken einfach **Ctrl**+**↑**+**F10** im Editor.

The screenshot shows the PyCharm IDE interface. A context menu is open over the file 'pi_liu_hui.py' in the editor. The menu includes options like 'New', 'Cut', 'Copy', 'Paste', 'Find Usages', 'Inspect Code...', 'Refactor', 'Bookmarks', 'Reformat Code', 'Optimize Imports', 'Delete...', 'Override File Type', 'Run 'pi_liu_hui'', 'Debug 'pi_liu_hui'', 'Modify Run Configuration...', 'Open in Right Split', 'Open In', 'Local History', 'Git', 'Repair IDE on File', and 'Reload from Disk'. The 'Run 'pi_liu_hui'' option is highlighted with a blue border. The code in the editor is a Python script for approximating pi using Liu Hui's method.

```
Liu Hui's Method to Approximate pi
Number of edges: We start with a hexagon, i.e., e=6.
de length: Initially 1, meaning the radius is also 1.
es, side length=s give us e * s / 2.

uble the number of edges...
rt(4 - (s ** 2)) # ...and recompute the side length.
es, side length=s give us e * s / 2.

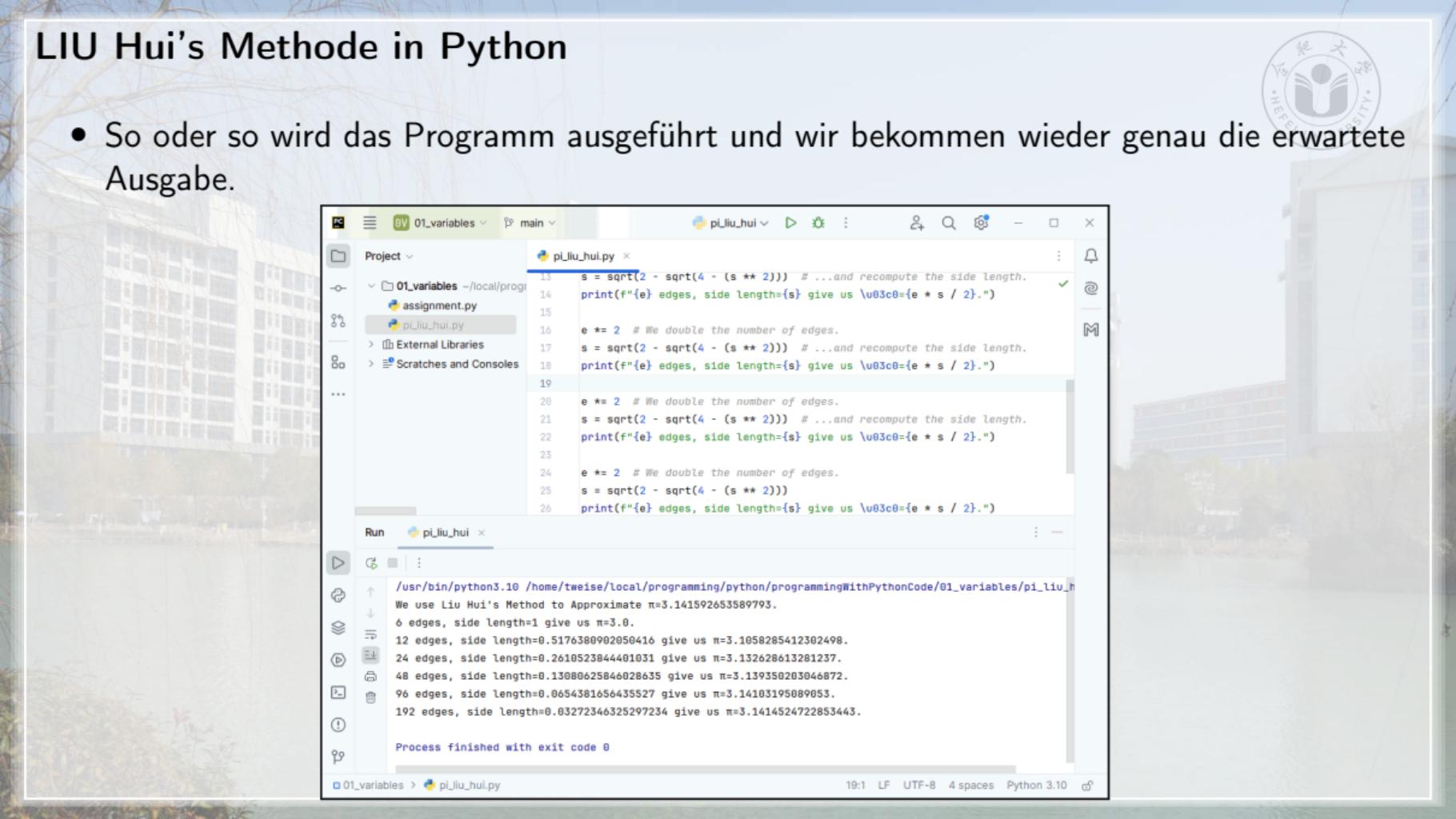
uble the number of edges...
rt(4 - (s ** 2)) # ...and recompute the side length.
es, side length=s give us e * s / 2.

uble the number of edges...
rt(4 - (s ** 2)) # ...and recompute the side length.
es, side length=s give us e * s / 2.

uble the number of edges...
rt(4 - (s ** 2)) # ...and recompute the side length.
es, side length=s give us e * s / 2.
```

LIU Hui's Methode in Python

- So oder so wird das Programm ausgeführt und wir bekommen wieder genau die erwartete Ausgabe.



A screenshot of a Python code editor showing the execution of a script named `pi_liu_hui.py`. The code implements Liu Hui's method to approximate the value of pi. It starts by calculating the side length for 6 edges, then iterates through 12, 24, 48, 96, and finally 192 edges, printing the resulting approximation each time. The output window shows the progression from 3.0 to 3.141592653589793.

```
s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.
print(f"{e} edges, side length={s} give us \u03c0=\u207ae * s / 2.)")

e *= 2 # We double the number of edges.
s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.
print(f"{e} edges, side length={s} give us \u03c0=\u207ae * s / 2.)"

e *= 2 # We double the number of edges.
s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.
print(f"{e} edges, side length={s} give us \u03c0=\u207ae * s / 2.)"

e *= 2 # We double the number of edges.
s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.
print(f"{e} edges, side length={s} give us \u03c0=\u207ae * s / 2.)"

e *= 2 # We double the number of edges.
s = sqrt(2 - sqrt(4 - (s ** 2))) # ...and recompute the side length.
print(f"{e} edges, side length={s} give us \u03c0=\u207ae * s / 2.)"

Process finished with exit code 0
```



Mehrfachzuweisungen und Wertetausch



Mehrfachzuweisungen und Wertetausch

- In Python können wir auch mehrere Werte zu mehreren Variablen zuweisen (aber natürlich jeweils einen Wert pro Variable).

```
1 a, b = 5, 10
2 print(f"{a = } , {b = }")
3
4 a, b = b, a
5 print(f"{a = } , {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = } , {y = } , {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = } , {y = } , {z = }")
```

Mehrfachzuweisungen und Wertetausch



- In Python können wir auch mehrere Werte zu mehreren Variablen zuweisen (aber natürlich jeweils einen Wert pro Variable).
- In diesem Fall trennen wir die Variablennamen und die Werte jeweils mit Kommas.

```
1 a, b = 5, 10
2 print(f"{a = } , {b = }")
3
4 a, b = b, a
5 print(f"{a = } , {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = } , {y = } , {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = } , {y = } , {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch



- In Python können wir auch mehrere Werte zu mehreren Variablen zuweisen (aber natürlich jeweils einen Wert pro Variable).
- In diesem Fall trennen wir die Variablennamen und die Werte jeweils mit Kommas.
- Die Zeile `a, b = 5, 10` ist äquivalent zu den beiden Zeilen `a = 5` und `b = 10`.

```
1 a, b = 5, 10
2 print(f"{a = } , {b = }")
3
4 a, b = b, a
5 print(f"{a = } , {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = } , {y = } , {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = } , {y = } , {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch



- In diesem Fall trennen wir die Variablennamen und die Werte jeweils mit Kommas.
- Die Zeile `a, b = 5, 10` ist äquivalent zu den beiden Zeilen `a = 5` und `b = 10`.
- Sie weist den Wert `5` der Variablen `a` zu und den Wert `10` der Variablen `b`.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch

- Die Zeile `a, b = 5, 10` ist äquivalent zu den beiden Zeilen `a = 5` und `b = 10`.
- Sie weist den Wert `5` der Variablen `a` zu und den Wert `10` der Variablen `b`.
- Danach gilt `a == 5` und `b == 10`.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch

- Sie weist den Wert 5 der Variablen `a` zu und den Wert 10 der Variablen `b`.
- Danach gilt `a == 5` und `b == 10`.
- `print(f"{a = }, {b = }")` druckt deshalb $a = 5, b = 10$ aus.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch



- Danach gilt `a == 5` and `b == 10`.
- `print(f"{a = }, {b = }")` druckt deshalb $a = 5, b = 10$ aus.
- Diese Methode kann auch verwendet werden, um die Werte von Variablen zu **tauschen**.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch



- `print(f"{a = }, {b = }")` druckt deshalb $a = 5, b = 10$ aus.
- Diese Methode kann auch verwendet werden, um die Werte von Variablen zu **tauschen**.
- `a, b = b, a` bedeutet im Grunde „der *neue* Wert von `a` wird der *jetzige* Wert von `b` und der *neue* Wert von `b` wird der *jetzige* Wert von `a`.“

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch



- Diese Methode kann auch verwendet werden, um die Werte von Variablen zu **tauschen**.
- `a, b = b, a` bedeutet im Grunde „der neue Wert von `a` wird der jetzige Wert von `b` und der neue Wert von `b` wird der jetzige Wert von `a`.“
- Die Zeile ist daher im Grunde äquivalent zu `t = a`, `a = b`, und `b = t`.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch



- Die Zeile ist daher im Grunde äquivalent zu `t = a`, `a = b`, und `b = t`.
- Ohne diese Doppel-Zuweisung müssten wir den Wert von `a` in einer temporären Variable `t` speichern, dann `a` mit `b` überschreiben, und dann den alten Wert von `a`, der jetzt in `t` wäre, in `b` speichern.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch



- Ohne diese Doppel-Zuweisung müssten wir den Wert von `a` in einer temporären Variable `t` speichern, dann `a` mit `b` überschreiben, und dann den alten Wert von `a`, der jetzt in `t` wäre, in `b` speichern.
- `a, b = b, a` schafft das in einem Schritt.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch

- `a, b = b, a` schafft das in einem Schritt.
- Und wir brauchen keine temporäre Variable.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch

- Und wir brauchen keine temporäre Variable.
- `print(f"{a = }, {b = }")` druckt nun $a = 10, b = 5$.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch

- `print(f"{}a = {} , {}b = {}")` druckt nun $a = 10, b = 5$.

Gute Praxis

Das Austauschen von Variablenwerten wird am Besten mit Hilfe von Mehrfachzuweisungen gemacht, z. B. `a, b = b, a`.



Mehrfachzuweisungen und Wertetausch

- Das selbe Konzept der Mehrfachzuweisung funktioniert auch mit mehreren Variablen.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch

- Das selbe Konzept der Mehrfachzuweisung funktioniert auch mit mehreren Variablen.
- `z, y, x = 1, 2, 3` speichert `1` in `z`, `2` in `y`, und `3` in `x`.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch



- Das selbe Konzept der Mehrfachzuweisung funktioniert auch mit mehreren Variablen.
- `z, y, x = 1, 2, 3` speichert `1` in `z`, `2` in `y`, und `3` in `x`.
- `print(f"{x = } , {y = } , {z = }")` druckt darum `x = 3, y = 2, z = 1`.

```
1 a, b = 5, 10
2 print(f"{a = } , {b = }")
3
4 a, b = b, a
5 print(f"{a = } , {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = } , {y = } , {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = } , {y = } , {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch

- `z, y, x = 1, 2, 3` speichert 1 in `z`, 2 in `y`, und 3 in `x`.
- `print(f"x = {x}, y = {y}, z = {z}")` druckt darum `x = 3, y = 2, z = 1`.
- Wir können auch mehrere Werte austauschen.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"x = {x}, y = {y}, z = {z}")
9
10 x, y, z = z, y, x
11 print(f"x = {x}, y = {y}, z = {z}")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch



- `print(f"\{x = \}, \{y = \}, \{z = \}")` druckt darum `x = 3, y = 2, z = 1.`
- Wir können auch mehrere Werte austauschen.
- `x, y, z = z, y, x` speichert den aktuellen Wert von `z` in `x`, den aktuellen Wert von `y` in `y`, und den aktuellen Wert von `x` in `z`.

```
1 a, b = 5, 10
2 print(f"\{a = \}, \{b = \}")
3
4 a, b = b, a
5 print(f"\{a = \}, \{b = \}")
6
7 z, y, x = 1, 2, 3
8 print(f"\{x = \}, \{y = \}, \{z = \}")
9
10 x, y, z = z, y, x
11 print(f"\{x = \}, \{y = \}, \{z = \}")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```

Mehrfachzuweisungen und Wertetausch



- Wir können auch mehrere Werte austauschen.
- `x, y, z = z, y, x` speichert den aktuellen Wert von `z` in `x`, den aktuellen Wert von `y` in `y`, und den aktuellen Wert von `x` in `z`.
- `print(f"{x = }, {y = }, {z = }")` ergibt nun also `x = 1, y = 2, z = 3`.

```
1 a, b = 5, 10
2 print(f"{a = }, {b = }")
3
4 a, b = b, a
5 print(f"{a = }, {b = }")
6
7 z, y, x = 1, 2, 3
8 print(f"{x = }, {y = }, {z = }")
9
10 x, y, z = z, y, x
11 print(f"{x = }, {y = }, {z = }")
```

```
1 a = 5, b = 10
2 a = 10, b = 5
3 x = 3, y = 2, z = 1
4 x = 1, y = 2, z = 3
```



Zusammenfassung



Zusammenfassung

- Wir haben nun gelernt, wie wir Werte in Variablen speichern können.





Zusammenfassung

- Wir haben nun gelernt, wie wir Werte in Variablen speichern können.
- Wir haben auch gelernt, dass wir diese Variablen dann ganz einfach anstelle der Werte verwenden können.

Zusammenfassung



- Wir haben nun gelernt, wie wir Werte in Variablen speichern können.
- Wir haben auch gelernt, dass wir diese Variablen dann ganz einfach anstelle der Werte verwenden können.
- Wir haben auch gelernt, dass wir Variablen mehrmals Werte zuweisen können.



Zusammenfassung

- Wir haben nun gelernt, wie wir Werte in Variablen speichern können.
- Wir haben auch gelernt, dass wir diese Variablen dann ganz einfach anstelle der Werte verwenden können.
- Wir haben auch gelernt, dass wir Variablen mehrmals Werte zuweisen können.
- Wir können also nun zum ersten Mal mehrzeilige Programme erstellen, in denen die Ergebnisse von einem Rechenschritt in den nächsten übernommen werden.

Zusammenfassung



- Wir haben nun gelernt, wie wir Werte in Variablen speichern können.
- Wir haben auch gelernt, dass wir diese Variablen dann ganz einfach anstelle der Werte verwenden können.
- Wir haben auch gelernt, dass wir Variablen mehrmals Werte zuweisen können.
- Wir können also nun zum ersten Mal mehrzeilige Programme erstellen, in denen die Ergebnisse von einem Rechenschritt in den nächsten übernommen werden.
- Das ist ein großer Sprung, denn nun kann unser Kode bereits wesentlich komplexer werden.

Zusammenfassung



- Wir haben nun gelernt, wie wir Werte in Variablen speichern können.
- Wir haben auch gelernt, dass wir diese Variablen dann ganz einfach anstelle der Werte verwenden können.
- Wir haben auch gelernt, dass wir Variablen mehrmals Werte zuweisen können.
- Wir können also nun zum ersten Mal mehrzeilige Programme erstellen, in denen die Ergebnisse von einem Rechenschritt in den nächsten übernommen werden.
- Das ist ein großer Sprung, denn nun kann unser Kode bereits wesentlich komplexer werden.
- Das führt dazu, dass wir uns um Dinge wie Kode-Qualität Sorgen machen müssen.

Zusammenfassung



- Wir haben nun gelernt, wie wir Werte in Variablen speichern können.
- Wir haben auch gelernt, dass wir diese Variablen dann ganz einfach anstelle der Werte verwenden können.
- Wir haben auch gelernt, dass wir Variablen mehrmals Werte zuweisen können.
- Wir können also nun zum ersten Mal mehrzeilige Programme erstellen, in denen die Ergebnisse von einem Rechenschritt in den nächsten übernommen werden.
- Das ist ein großer Sprung, denn nun kann unser Kode bereits wesentlich komplexer werden.
- Das führt dazu, dass wir uns um Dinge wie Kode-Qualität Sorgen machen müssen.
- Wir wollen lesbaren Kode schreiben und halten uns daher an Stilvorgaben.

Zusammenfassung



- Wir haben nun gelernt, wie wir Werte in Variablen speichern können.
- Wir haben auch gelernt, dass wir diese Variablen dann ganz einfach anstelle der Werte verwenden können.
- Wir haben auch gelernt, dass wir Variablen mehrmals Werte zuweisen können.
- Wir können also nun zum ersten Mal mehrzeilige Programme erstellen, in denen die Ergebnisse von einem Rechenschritt in den nächsten übernommen werden.
- Das ist ein großer Sprung, denn nun kann unser Kode bereits wesentlich komplexer werden.
- Das führt dazu, dass wir uns um Dinge wie Kode-Qualität Sorgen machen müssen.
- Wir wollen lesbaren Kode schreiben und halten uns daher an Stilvorgaben.
- Wir kommentieren unseren Kode, so dass wir (und andere) später noch verstehen können, was wir uns beim Programmieren gedacht haben.

Zusammenfassung



- Wir haben nun gelernt, wie wir Werte in Variablen speichern können.
- Wir haben auch gelernt, dass wir diese Variablen dann ganz einfach anstelle der Werte verwenden können.
- Wir haben auch gelernt, dass wir Variablen mehrmals Werte zuweisen können.
- Wir können also nun zum ersten Mal mehrzeilige Programme erstellen, in denen die Ergebnisse von einem Rechenschritt in den nächsten übernommen werden.
- Das ist ein großer Sprung, denn nun kann unser Kode bereits wesentlich komplexer werden.
- Das führt dazu, dass wir uns um Dinge wie Kode-Qualität Sorgen machen müssen.
- Wir wollen lesbaren Kode schreiben und halten uns daher an Stilvorgaben.
- Wir kommentieren unseren Kode, so dass wir (und andere) später noch verstehen können, was wir uns beim Programmieren gedacht haben.
- Und damit können wir nun auch bereits sinnvolle und coole Berechnungen anstellen!

Zusammenfassung



- Wir haben nun gelernt, wie wir Werte in Variablen speichern können.
- Wir haben auch gelernt, dass wir diese Variablen dann ganz einfach anstelle der Werte verwenden können.
- Wir haben auch gelernt, dass wir Variablen mehrmals Werte zuweisen können.
- Wir können also nun zum ersten Mal mehrzeilige Programme erstellen, in denen die Ergebnisse von einem Rechenschritt in den nächsten übernommen werden.
- Das ist ein großer Sprung, denn nun kann unser Kode bereits wesentlich komplexer werden.
- Das führt dazu, dass wir uns um Dinge wie Kode-Qualität Sorgen machen müssen.
- Wir wollen lesbaren Kode schreiben und halten uns daher an Stilvorgaben.
- Wir kommentieren unseren Kode, so dass wir (und andere) später noch verstehen können, was wir uns beim Programmieren gedacht haben.
- Und damit können wir nun auch bereits sinnvolle und coole Berechnungen anstellen!
- Mit Mehrfachzuweisungen können wir mehrere Variablen gleichzeitig belegen und auch Variablenwerte austauschen.



谢谢您们！
Thank you!
Vielen Dank!

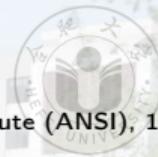


References I



- [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. 219, 220).
- [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. 219).
- [3] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 218).
- [4] Florian Bruhin. *Python f-Strings*. Winterthur, Switzerland: Bruhin Software, 31. Mai 2023. URL: <https://fstring.help> (besucht am 2024-07-25) (siehe S. 218).
- [5] Antonio Cavacini. "Is the CE/BCE notation becoming a standard in scholarly literature?" *Scientometrics* 102(2):1661–1668, Juli 2015. London, England, UK: Springer Nature Limited. ISSN: 0138-9130. doi:10.1007/s11192-014-1352-1 (siehe S. 218).
- [6] Josh Centers. *Take Control of iOS 18 and iPadOS 18*. San Diego, CA, USA: Take Control Books, Dez. 2024. ISBN: 978-1-990783-55-5 (siehe S. 219).
- [7] Donald D. Chamberlin. "50 Years of Queries". *Communications of the ACM (CACM)* 67(8):110–121, Aug. 2024. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/3649887. URL: <https://cacm.acm.org/research/50-years-of-queries> (besucht am 2025-01-09) (siehe S. 220).
- [8] 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. 220).
- [9] Edgar Frank „Ted“ Codd. "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM (CACM)* 13(6):377–387, Juni 1970. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/362384.362685. URL: <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf> (besucht am 2025-01-05) (siehe S. 219).
- [10] Christopher Cullen. "Learning from Liu Hui? A Different Way to Do Mathematics". *Notices of the American Mathematical Society* 49(7):783–790, Aug. 2002. Providence, RI, USA: American Mathematical Society (AMS). ISSN: 1088-9477. URL: <https://www.ams.org/notices/200207/comm-cullen.pdf> (besucht am 2024-08-09) (siehe S. 110–114).

References II



- [11] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 220).
- [12] Joseph W. Dauben. "Archimedes and Liu Hui on Circles and Spheres". *Ontology Studies (Cuadernos de Ontología)* 10:21–38, 2010. Leioa, Bizkaia, Spain: Universidad del País Vasco / Euskal Herriko Unibertsitatea. ISSN: 1576-2270. URL: <https://ddd.uab.cat/pub/ontstu/15762270n10/15762270n10p21.pdf> (besucht am 2024-08-10) (siehe S. 110–114).
- [13] Matt David und Blake Barnhill. *How to Teach People SQL*. San Francisco, CA, USA: The Data School, Chart.io, Inc., 10. Dez. 2019–10. Apr. 2023. URL: <https://dataschool.com/how-to-teach-people-sql> (besucht am 2025-02-27) (siehe S. 220).
- [14] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 220).
- [15] Luca Ferrari und Enrico Pirozzi. *Learn PostgreSQL*. 2. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Okt. 2023. ISBN: 978-1-83763-564-1 (siehe S. 219).
- [16] Michael Filaseta. "The Transcendence of e and π ". In: *Math 785: Transcendental Number Theory*. Columbia, SC, USA: University of South Carolina, Frühling 2011. Kap. 6. URL: <https://people.math.sc.edu/filaseta/gradcourses/Math785/Math785Notes6.pdf> (besucht am 2024-07-05) (siehe S. 221).
- [17] "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. 218).
- [18] 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. 218).
- [19] 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. 218).

References III



- [20] Terry Halpin und Tony Morgan. *Information Modeling and Relational Databases*. 3. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juli 2024. ISBN: [978-0-443-23791-1](#) (siehe S. 219).
- [21] Jan L. Harrington. *Relational Database Design and Implementation*. 4. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Apr. 2016. ISBN: [978-0-12-849902-3](#) (siehe S. 219).
- [22] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: [978-1-0981-0894-6](#) (siehe S. 219).
- [23] 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. 220).
- [24] 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. 219).
- [25] *Information Technology – Database Languages – SQL – Part 1: Framework (SQL/Framework), Part 1*. International Standard ISO/IEC 9075-1:2023(E), Sixth Edition, (ANSI X3.135). Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Juni 2023. URL: [https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_\(en\).zip](https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_(en).zip) (besucht am 2025-01-08). Consists of several parts, see <https://modern-sql.com/standard> for information where to obtain them. (Siehe S. 220).
- [26] *Information Technology – Universal Coded Character Set (UCS)*. International Standard ISO/IEC 10646:2020. Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Dez. 2020 (siehe S. 221).
- [27] Arthur Jones, Kenneth R. Pearson und Sidney A. Morris. "Transcendence of e and π ". In: *Abstract Algebra and Famous Impossibilities*. Universitext (UTX). New York, NY, USA: Springer New York, 1991. Kap. 9, S. 115–161. ISSN: [0172-5939](#). ISBN: [978-1-4419-8552-1](#). doi:[10.1007/978-1-4419-8552-1_8](#) (siehe S. 221).
- [28] ".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. 220).

References IV



- [29] Shen Kangshen, John Newsome Crossley und Anthony W.-C. Lun. *The Nine Chapters on the Mathematical Art: Companion and Commentary*. Oxford, Oxfordshire, England, UK: Oxford University Press, 7. Okt. 1999. ISBN: 978-0-19-853936-0.
doi:10.1093/oso/9780198539360.001.0001 (siehe S. 110–114).
- [30] 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. 219).
- [31] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 219).
- [32] "Mathematical Functions and Operators". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.3. URL: <https://www.postgresql.org/docs/17/functions-math.html> (besucht am 2025-02-27) (siehe S. 221).
- [33] 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. 218).
- [34] Jim Melton und Alan R. Simon. *SQL: 1999 – Understanding Relational Language Components*. The Morgan Kaufmann Series in Data Management Systems. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juni 2001. ISBN: 978-1-55860-456-8 (siehe S. 220).
- [35] 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. 218).
- [36] Ivan Niven. "The Transcendence of π ". *The American Mathematical Monthly* 46(8):469–471, Okt. 1939. London, England, UK: Taylor and Francis Ltd. ISSN: 1930-0972. doi:10.2307/2302515 (siehe S. 221).
- [37] John J. O'Connor und Edmund F. Robertson. *Liu Hui*. St Andrews, Scotland, UK: University of St Andrews, School of Mathematics and Statistics, Dez. 2003. URL: https://mathshistory.st-andrews.ac.uk/Biographies/Liu_Hui (besucht am 2024-08-10) (siehe S. 110–114).
- [38] Regina O. Obe und Leo S. Hsu. *PostgreSQL: Up and Running*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Okt. 2017. ISBN: 978-1-4919-6336-4 (siehe S. 219).
- [39] *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), Feb. 2025. URL: <https://www.postgresql.org/docs/17/index.html> (besucht am 2025-02-25).

References V



- [40] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 219).
- [41] Ernest E. Rothman, Rich Rosen und Brian Jepson. *Mac OS X for Unix Geeks*. 4. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2008. ISBN: 978-0-596-52062-5 (siehe S. 219).
- [42] Ahmad Sahar. *iOS 26 Programming for Beginners*. 10. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Nov. 2025. ISBN: 978-1-80602-393-6 (siehe S. 221).
- [43] Syamal K. Sen und Ravi P. Agarwal. "Existence of year zero in astronomical counting is advantageous and preserves compatibility with significance of AD, BC, CE, and BCE". In: *Zero – A Landmark Discovery, the Dreadful Void, and the Ultimate Mind*. Amsterdam, The Netherlands: Elsevier B.V., 2016. Kap. 5.5, S. 94–95. ISBN: 978-0-08-100774-7. doi:10.1016/C2015-0-02299-7 (siehe S. 218).
- [44] 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. 219).
- [45] Bryan Sills, Brian Gardner, Kristin Marsicano und Chris Stewart. *Android Programming: The Big Nerd Ranch Guide*. 5. Aufl. Reading, MA, USA: Addison-Wesley Professional, Mai 2022. ISBN: 978-0-13-764579-4 (siehe S. 218).
- [46] Drew Smith. *Modern Apple Platform Administration – macOS and iOS Essentials* (2025). Birmingham, England, UK: Packt Publishing Ltd, Feb. 2025. ISBN: 978-1-80580-309-6 (siehe S. 219).
- [47] 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. 218).
- [48] John Miles Smith und Philip Yen-Tang Chang. "Optimizing the Performance of a Relational Algebra Database Interface". *Communications of the ACM (CACM)* 18(10):568–579, Okt. 1975. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/361020.361025 (siehe S. 219).
- [49] "SQL Commands". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Part VI. Reference. URL: <https://www.postgresql.org/docs/17/sql-commands.html> (besucht am 2025-02-25) (siehe S. 220).

References VI



- [50] Ryan K. Stephens und Ronald R. Plew. *Sams Teach Yourself SQL in 21 Days*. 4. Aufl. Sams Tech Yourself. Indianapolis, IN, USA: SAMS Technical Publishing und Hoboken, NJ, USA: Pearson Education, Inc., Okt. 2002. ISBN: 978-0-672-32451-2 (siehe S. 215, 220).
- [51] Ryan K. Stephens, Ronald R. Plew, Bryan Morgan und Jeff Perkins. *SQL in 21 Tagen. Die Datenbank-Abfragesprache SQL vollständig erklärt (in 14/21 Tagen)*. 6. Aufl. Burgthann, Bayern, Germany: Markt+Technik Verlag GmbH, Feb. 1998. ISBN: 978-3-8272-2020-2. Translation of⁵⁰ (siehe S. 220).
- [52] Philip D. Straffin Jr. "Liu Hui and the First Golden Age of Chinese Mathematics". *Mathematics Magazine* 71(3):163–181, Juni 1998. London, England, UK: Taylor and Francis Ltd. ISSN: 0025-570X. doi:10.2307/2691200. URL: <https://www.researchgate.net/publication/237334342> (besucht am 2024-08-10) (siehe S. 110–114).
- [53] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 219, 220).
- [54] Alkin Tezusyal und Ibrar Ahmed. *Database Design and Modeling with PostgreSQL and MySQL*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2024. ISBN: 978-1-80323-347-5 (siehe S. 219).
- [55] *The Unicode Standard, Version 15.1: Archived Code Charts*. South San Francisco, CA, USA: The Unicode Consortium, 25. Aug. 2023. URL: <https://www.unicode.org/Public/15.1.0/charts/CodeCharts.pdf> (besucht am 2024-07-26) (siehe S. 221).
- [56] 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. 219).
- [57] *Unicode®15.1.0*. South San Francisco, CA, USA: The Unicode Consortium, 12. Sep. 2023. ISBN: 978-1-936213-33-7. URL: <https://www.unicode.org/versions/Unicode15.1.0> (besucht am 2024-07-26) (siehe S. 221).
- [58] Bruce M. Van Horn II und Quan Nguyen. *Hands-On Application Development with PyCharm*. 2. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Okt. 2023. ISBN: 978-1-83763-235-0 (siehe S. 219).



References VII

- [59] 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, 81–86, 97–102).
- [60] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 219).
- [61] Thomas Weise (汤卫思). *Databases*. 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), 2025. URL: <https://thomasweise.github.io/databases> (besucht am 2025-01-05) (siehe S. 218, 219).
- [62] 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. 219).
- [63] *What is a Relational Database?* Armonk, NY, USA: International Business Machines Corporation (IBM), 20. Okt. 2021–12. Dez. 2024. URL: <https://www.ibm.com/think/topics/relational-databases> (besucht am 2025-01-05) (siehe S. 219).
- [64] Kevin Wilson. *Python Made Easy*. Birmingham, England, UK: Packt Publishing Ltd, Aug. 2024. ISBN: 978-1-83664-615-0 (siehe S. 219).
- [65] Martin Yanev. *PyCharm Productivity and Debugging Techniques*. Birmingham, England, UK: Packt Publishing Ltd, Okt. 2022. ISBN: 978-1-83763-244-2 (siehe S. 219).
- [66] Kinza Yasar und Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., Juni 2024. URL: <https://www.techtarget.com/searchdatamanagement/definition/database-management-system> (besucht am 2025-01-11) (siehe S. 218).
- [67] Wenqi Ying (应雯棋), Hrsg. *Commemoration of Ancient Chinese Mathematical Master Liu Hui for his Timeless Influence on Mathematics and Civilizational Exchange*. Bd. 48 (Special Issue) der Reihe CAST Newsletter. China, Beijing (中国北京市): 中国科学技术协会 (China Association for Science and Technology, CAST), Nov. 2024. URL: https://english.cast.org.cn/cms_files/filemanager/1941250207/attach/202412/8f23655a82364d19ad7874eb37b23035.pdf (besucht am 2025-08-24). Proofreader: Yumeng Wei (魏雨萌), Designer: Shan Zhang (张珊) (siehe S. 110–114).

References VIII



- [68] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 218).
- [69] Nicola Abdo Ziadeh, Michael B. Rowton, A. Geoffrey Woodhead, Wolfgang Helck, Jean L.A. Filliozat, Hiroyuki Momo, Eric Thompson, E.J. Wiesenberg und Shih-ch'ang Wu. "Chronology – Christian History, Dates, Events". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 26. Juli 1999–20. März 2024. URL: <https://www.britannica.com/topic/chronology/Christian> (besucht am 2025-08-27) (siehe S. 218).



Glossary (in English) I

Android is a common operating system for mobile phones⁴⁵.

Bash is 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^{3,35,68}. Learn more at <https://www.gnu.org/software/bash>.

BCE The time notation *before Common Era* is a non-religious but chronological equivalent alternative to the traditional *Before Christ (BC)* notation, which refers to the years *before* the birth of Jesus Christ⁵. The years BCE are counted down, i.e., the larger the year, the farther in the past. The year 1 BCE comes directly before the year 1 CE^{43,69}.

CE The time notation *Common Era* is a non-religious but chronological equivalent alternative to the traditional *Anno Domini (AD)* notation, which refers to the years *after* the birth of Jesus Christ⁵. The years CE are counted upwards, i.e., the smaller they are, the farther they are in the past. The year 1 CE comes directly after the year 1 before Common Era (BCE)^{43,69}.

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*⁶¹.

DBMS A *database management system* is the software layer located between the user or application and the database (DB). The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB⁶⁶.

f-string let you include the results of expressions in strings^{4,17–19,33,47}. They can contain expressions (in curly braces) like `f"{}{}` that are then transformed to text via (string) interpolation, which turns the string to `"a5b"`. F-strings are delimited by `f"..."`.

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.

Glossary (in English) II



iOS is the operating system that powers Apple iPhones^{6,46}. Learn more at <https://www.apple.com/ios>.

Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows^{1,22,44,56,60}. 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^{41,46}. Learn more at <https://www.apple.com/macos>.

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>.

OS Operating System, the system that runs your computer, see, e.g., Linux, Microsoft Windows, macOS, and Android.

PostgreSQL An open source object-relational database management system (DBMS)^{15,38,40,54}. See <https://postgresql.org> for more information.

PyCharm is the convenient Python IDE that we recommend for this course^{58,64,65}. 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>.

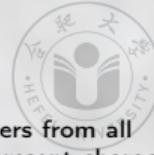
Python The Python programming language^{24,30,31,62}, i.e., what you will learn about in our book⁶². Learn more at <https://python.org>.

relational database A relational DB is a database that organizes data into rows (tuples, records) and columns (attributes), which collectively form tables (relations) where the data points are related to each other^{9,20,21,48,53,61,63}.



Glossary (in English) III

- SQL** The *Structured Query Language* is basically a programming language for querying and manipulating relational databases^{7,11,13,14,25,34,49–51,53}. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference⁴⁹.
- 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`)²⁸. 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`)²⁸. 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`)²⁸. 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^{1,8}. 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 `Windows`+`R`, dann Schreiben von `cmd`, dann Druck auf `Enter`. 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^{8,23}. 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) IV

- Unicode A standard for assigning characters to numbers^{26,55,57}. The Unicode standard supports basically all characters from all languages that are currently in use, as well as many special symbols. It is the predominantly used way to represent characters in computers and is regularly updated and improved.
- Xcode is offers the tools for developing, testing, and distributing applications as well as an IDE for Apple platforms such as macOS and iOS⁴².
- π is the ratio of the circumference U of a circle and its diameter d , i.e., $\pi = U/d$. $\pi \in \mathbb{R}$ is an irrational and transcendental number^{16,27,36}, which is approximately $\pi \approx 3.141\,592\,653\,589\,793\,238\,462\,643$. In Python, it is provided by the `math` module as constant `pi` with value `3.141592653589793`. In PostgreSQL, it is provided by the SQL function `pi()` with value `3.141592653589793`³².
- \mathbb{R} the set of the real numbers.