合肥大學
HEFEI UNIVERSITY




# Programming with Python

# 11. Der Datentyp str

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn · https://thomasweise.github.io

School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2026-11-04

# 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. Grundlegende Operationen
3. Die Funktion str und f-Strings
4. Strings in Andere Datentypen Konvertieren
5. Strings Escaping
6. Mehrzeilige Strings
7. Unicode
8. Zusammenfassung

# Einleitung

# Einleitung

- Der vierte wichtige grundlegende Datentyp in Python sind Zeichenketten, Texte, auch genannt *Strings*.[60]

# Einleitung

- Der vierte wichtige grundlegende Datentyp in Python sind Zeichenketten, Texte, auch genannt *Strings*.[60]
- Zeichenketten sind beliebig lange Sequenzen von Text-Zeichen.

# Einleitung



- Der vierte wichtige grundlegende Datentyp in Python sind Zeichenketten, Texte, auch genannt *Strings*.[60]
- Zeichenketten sind beliebig lange Sequenzen von Text-Zeichen.
- In Python sind sie durch den Datentyp `str` repräsentiert.

# Einleitung



- Der vierte wichtige grundlegende Datentyp in Python sind Zeichenketten, Texte, auch genannt *Strings*.[60]
- Zeichenketten sind beliebig lange Sequenzen von Text-Zeichen.
- In Python sind sie durch den Datentyp `str` repräsentiert.
- Wir haben sie bereits oftmals implizit oder explizit genutzt, z. B. in unserem ersten Programm, das einfach `"Hello World"` ausgedruckt hat.

# Einleitung



- Der vierte wichtige grundlegende Datentyp in Python sind Zeichenketten, Texte, auch genannt *Strings*.[60]
- Zeichenketten sind beliebig lange Sequenzen von Text-Zeichen.
- In Python sind sie durch den Datentyp `str` repräsentiert.
- Wir haben sie bereits oftmals implizit oder explizit genutzt, z. B. in unserem ersten Programm, das einfach `"Hello World"` ausgedruckt hat.
- `"Hello World"` is so ein Text-String.

# Grundlegende Operationen



## Strings Definieren, Verketten, und Indizieren



- Schauen wir uns die grundlegenden Operationen für Strings an.

# Strings Definieren, Verketten, und Indizieren

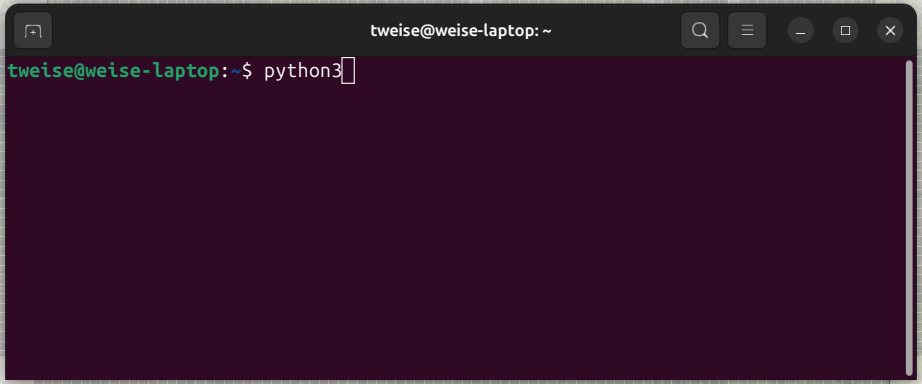
- Wir öffnen ein Terminal (Unter Ubuntu Linux durch Drücken von Ctrl + Alt + T, unter Microsoft Windows durch Druck auf ⊞ + R, dann Schreiben von `cmd`, dann Druck auf ↵.)



# Strings Definieren, Verketten, und Indizieren

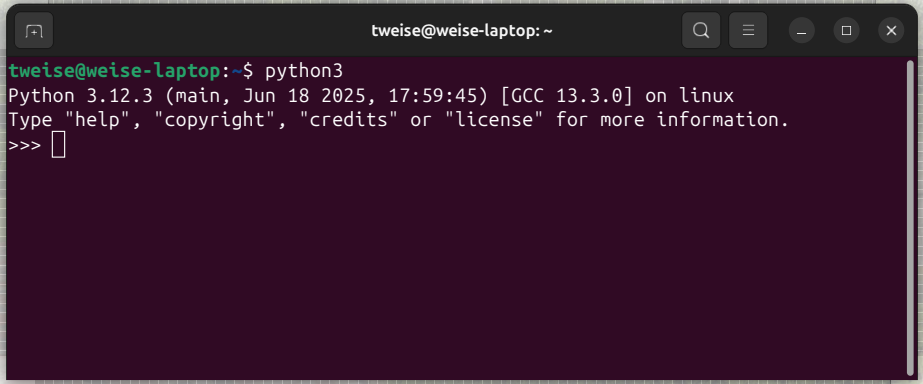


- Wir schreiben `python3` und drücken ↵.



# Strings Definieren, Verketten, und Indizieren

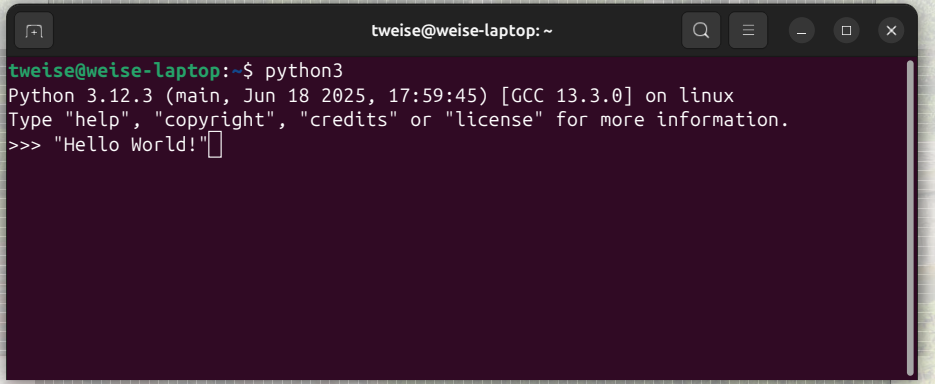
- Der Python-Interpreter startet.



# Strings Definieren, Verketten, und Indizieren

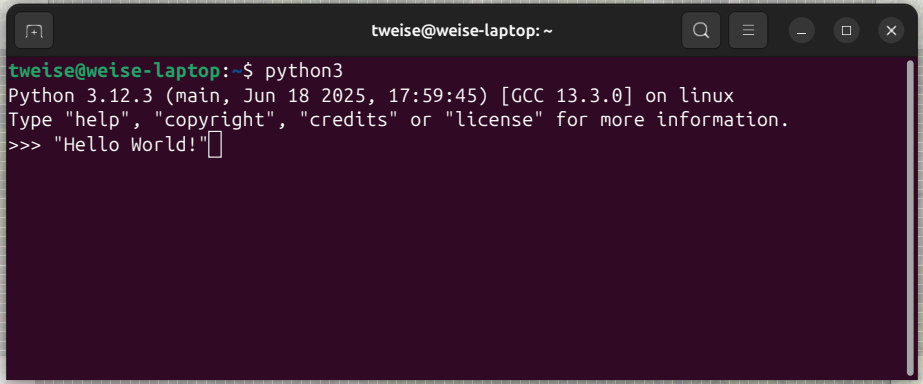


- Es gibt zwei grundlegende Methoden, Strings zu definieren.



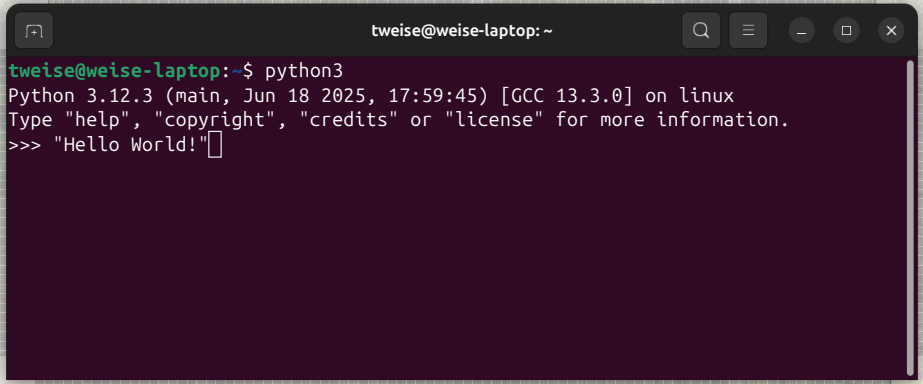
# Strings Definieren, Verketten, und Indizieren

- Es gibt zwei grundlegende Methoden, Strings zu definieren. Entweder mit einfachen oder doppelten Anführungszeichen.



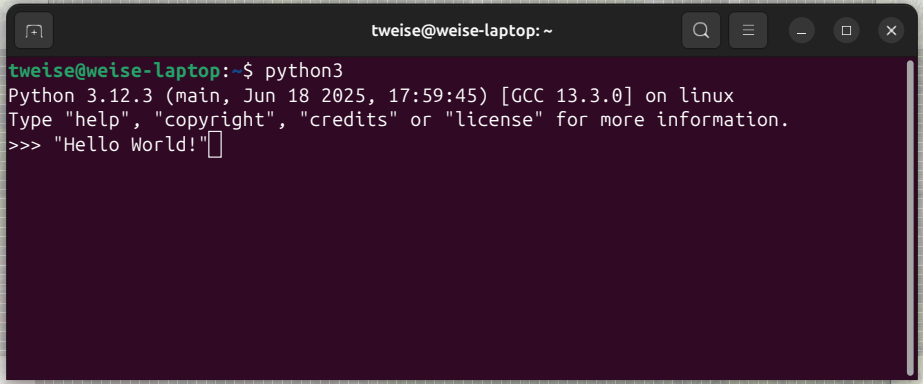
# Strings Definieren, Verketten, und Indizieren

- Es gibt zwei grundlegende Methoden, Strings zu definieren. Entweder mit einfachen oder doppelten Anführungszeichen. Probieren wir es zuerst mit den doppelten Anführungszeichen und schreiben `"Hello World!"`.



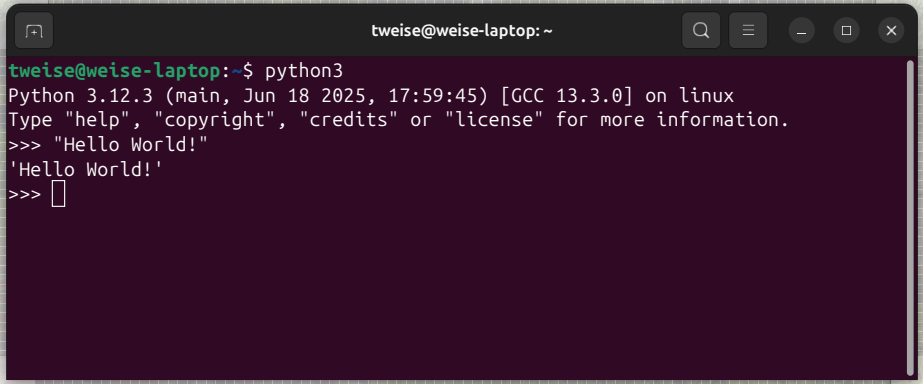
# Strings Definieren, Verketten, und Indizieren

- Es gibt zwei grundlegende Methoden, Strings zu definieren. Probieren wir es zuerst mit den doppelten Anführungszeichen und schreiben `"Hello World!"`. Beachten Sie, dass die Anführungszeichen selbst nicht zum String gehören, sondern ihn nur begrenzen.



# Strings Definieren, Verketten, und Indizieren

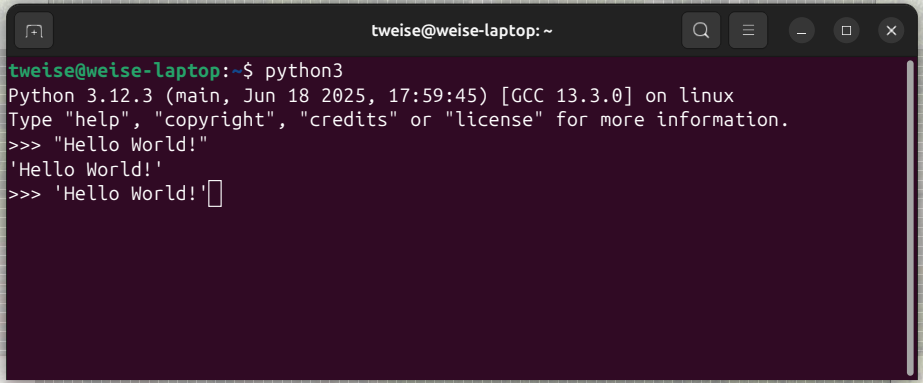
- Es gibt zwei grundlegende Methoden, Strings zu definieren. Probieren wir es zuerst mit den doppelten Anführungszeichen und schreiben `"Hello World!"`. Beachten Sie, dass die Anführungszeichen selbst nicht zum String gehören, sondern ihn nur begrenzen. Der Text wird uns wieder ausgegeben.



# Strings Definieren, Verketten, und Indizieren

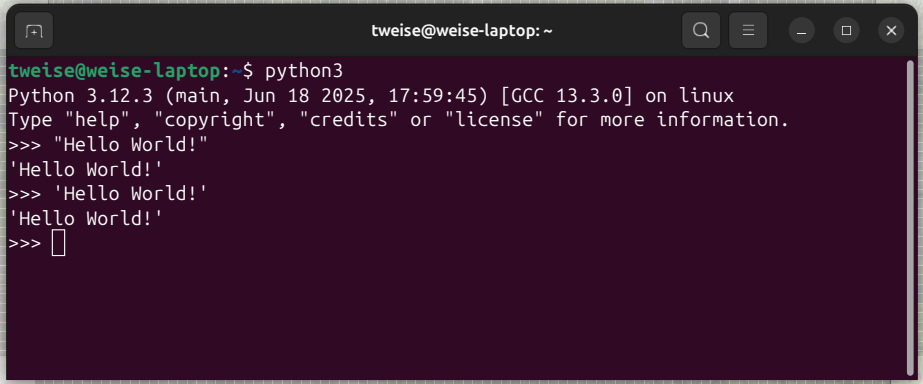


- Nun probieren wir es mit einfachen Anführungszeichen und schreiben `'Hello World!'`.



# Strings Definieren, Verketten, und Indizieren

- Nun probieren wir es mit einfachen Anführungszeichen und schreiben `'Hello World!'`. Und das wird uns auch wieder ausgegeben.



## Strings Definieren, Verketten, und Indizieren



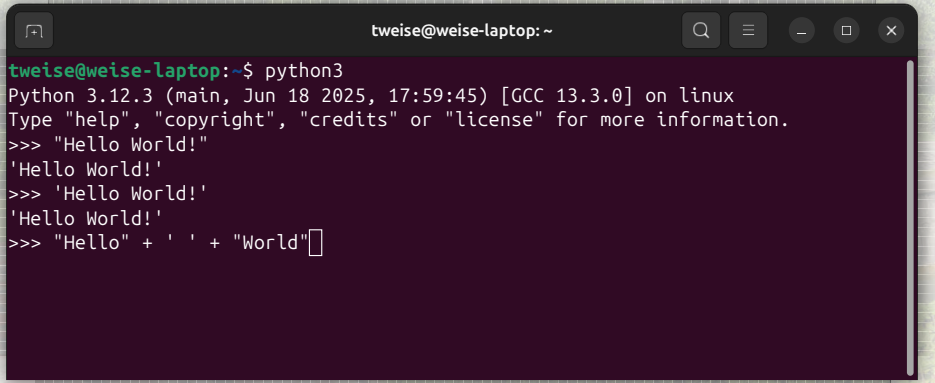
- Es gibt zwei grundlegende Methoden, Strings zu definieren. Probieren wir es zuerst mit den doppelten Anführungszeichen und schreiben `"Hello World!"`. Beachten Sie, dass die Anführungszeichen selbst nicht zum String gehören, sondern ihn nur begrenzen. Der Text wird uns wieder ausgegeben.
- Nun probieren wir es mit einfachen Anführungszeichen und schreiben `'Hello World!'`. Und das wird uns auch wieder ausgegeben.

**Gute Praxis**

Beim Definieren von String-Literalen sollte die Variante mit doppelten Anführungszeichen (`"..."`) bevorzugt werden. (Der *Style Guide for Python Code*[69] gibt keine Empfehlung, aber vielleicht für Konsistenz mit den *Docstring Conventions*[27].)

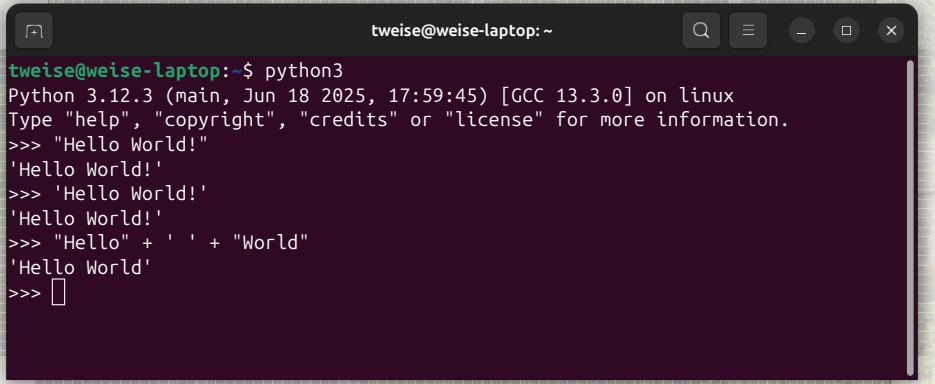
# Strings Definieren, Verketten, und Indizieren

- Strings können mit `+` aneinander angehängt (verkettet) werden.



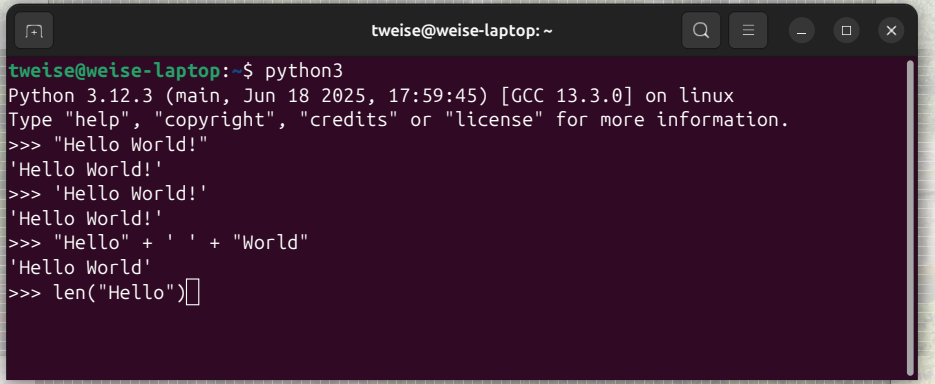
# Strings Definieren, Verketten, und Indizieren

- Strings können mit `+` aneinander angehängt (verkettet) werden. Sie ergeben dann ein einzigen String, der aus allen Teilstrings in der richtigen Reihenfolge besteht.



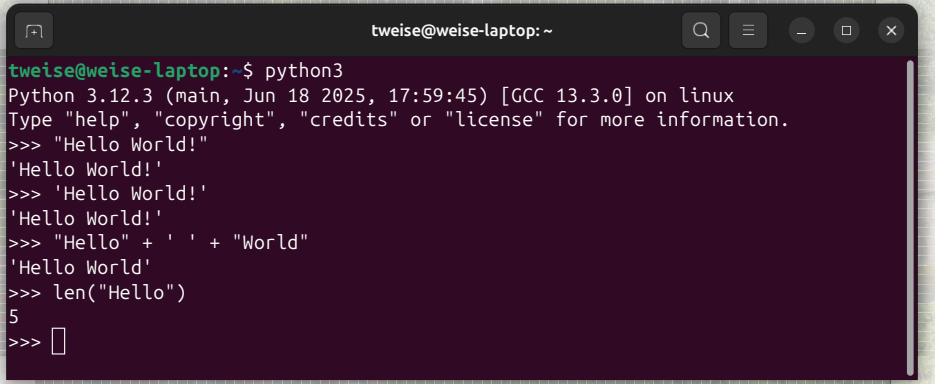
# Strings Definieren, Verketten, und Indizieren

- Die Funktion `len(x)` liefert uns die Länge des Strings `x`, also die Anzahl der Zeichen in der Zeichenkette.



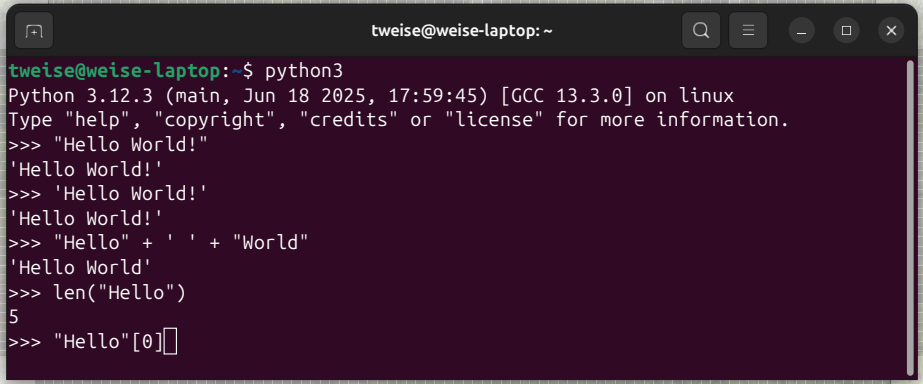
# Strings Definieren, Verketten, und Indizieren

- Die Funktion `len(x)` liefert uns die Länge des Strings `x`, also die Anzahl der Zeichen in der Zeichenkette. „Hello“ besteht aus fünf Zeichen.



# Strings Definieren, Verketten, und Indizieren

- Wir können auch einzelne Zeichen aus einem String herausholen. `x[i]` liefert das `i` + 1-te Zeichen.



# Strings Definieren, Verketten, und Indizieren

- Wir können auch einzelne Zeichen aus einem String herausholen. `x[i]` liefert das `i` + 1-te Zeichen. Das erste Zeichen ist also an Index 0, bei „Hello“ ist das „H“. Dieses Zeichen ist natürlich wieder ein `str` (nur mit Länge 1).



# Strings Definieren, Verketten, und Indizieren

- Das zweite Zeichen bekommen wir durch `[1]`.



# Strings Definieren, Verketten, und Indizieren



- Das zweite Zeichen bekommen wir durch `[1]`. Bei „Hello“ ist das „e“.

tweise@weise-laptop: ~

```
>>> "Hello World!"
'Hello World!'
>>> 'Hello World!'
'Hello World!'
>>> "Hello" + ' ' + "World"
'Hello World'
>>> len("Hello")
5
>>> "Hello"[0]
'H'
>>> "Hello"[1]
'e'
>>>
```

# Strings Definieren, Verketten, und Indizieren



- Das dritte Zeichen bekommen wir durch `[2]`.

tweise@weise-laptop: ~

```
>>> "Hello World!"
'Hello World!'
>>> 'Hello World!'
'Hello World!'
>>> "Hello" + ' ' + "World"
'Hello World'
>>> len("Hello")
5
>>> "Hello"[0]
'H'
>>> "Hello"[1]
'e'
>>> "Hello"[2]
```

# Strings Definieren, Verketten, und Indizieren



- Das dritte Zeichen bekommen wir durch `[2]`. Bei „Hello“ ist das „l“.

tweise@weise-laptop: ~

```
>>> 'Hello World!'
'Hello World!'
>>> "Hello" + ' ' + "World"
'Hello World'
>>> len("Hello")
5
>>> "Hello"[0]
'H'
>>> "Hello"[1]
'e'
>>> "Hello"[2]
'l'
>>>
```

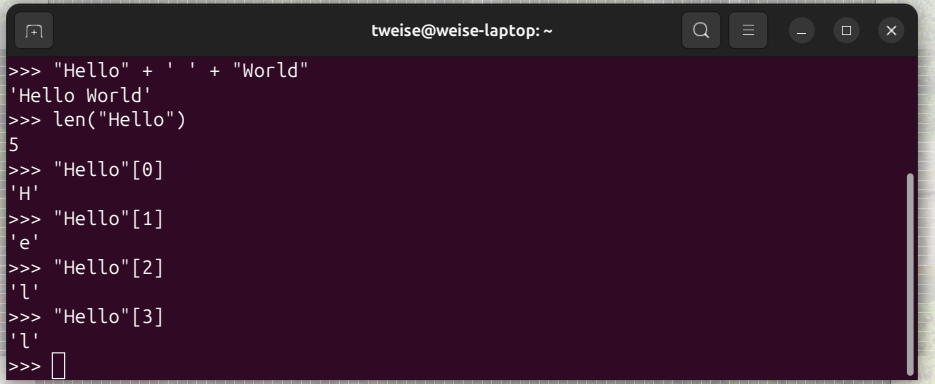
# Strings Definieren, Verketten, und Indizieren

- Das vierte Zeichen bekommen wir durch `[3]`.



# Strings Definieren, Verketten, und Indizieren

- Das vierte Zeichen bekommen wir durch `[3]`. Bei „Hello“ ist das „l“.



# Strings Definieren, Verketten, und Indizieren



- Das fünfte Zeichen bekommen wir durch `[4]`.

tweise@weise-laptop: ~

```
>>> "Hello" + ' ' + "World"
'Hello World'
>>> len("Hello")
5
>>> "Hello"[0]
'H'
>>> "Hello"[1]
'e'
>>> "Hello"[2]
'l'
>>> "Hello"[3]
'l'
>>> "Hello"[4]
```

# Strings Definieren, Verketten, und Indizieren



- Das fünfte Zeichen bekommen wir durch `[4]`. Bei „Hello“ ist das „o“.

tweise@weise-laptop: ~

```
>>> len("Hello")
5
>>> "Hello"[0]
'H'
>>> "Hello"[1]
'e'
>>> "Hello"[2]
'l'
>>> "Hello"[3]
'l'
>>> "Hello"[4]
'o'
>>>
```

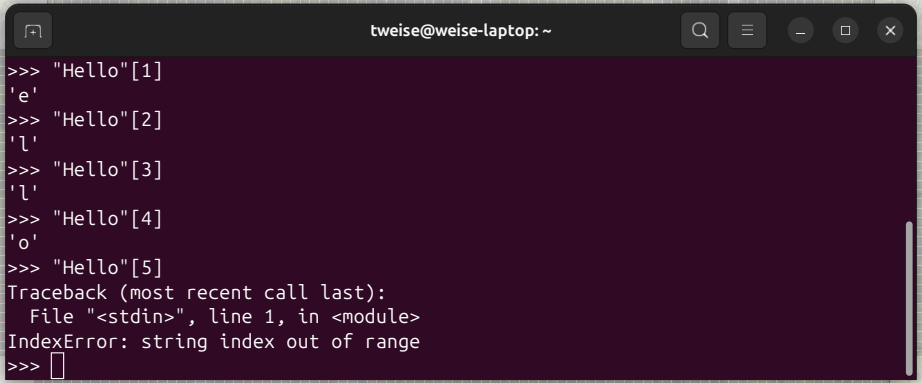
# Strings Definieren, Verketten, und Indizieren

- „Hello“ hat fünf Zeichen. Was passiert, wenn wir versuchen, auf das sechste Zeichen via `[5]` zuzugreifen?



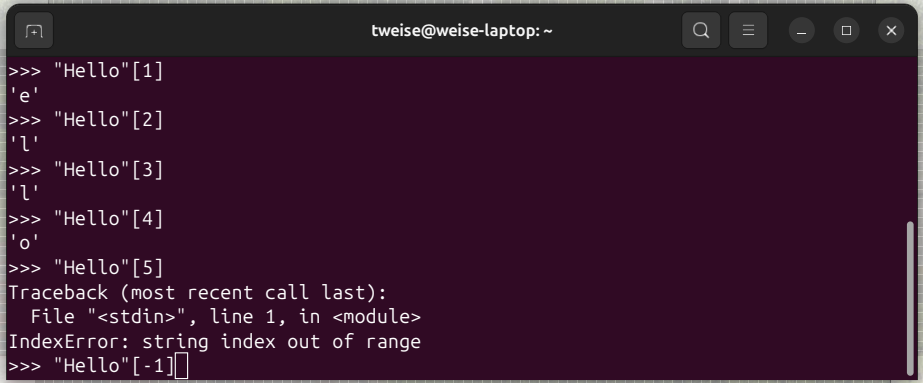
# Strings Definieren, Verketten, und Indizieren

- „Hello“ hat fünf Zeichen. Was passiert, wenn wir versuchen, auf das sechste Zeichen via `[5]` zuzugreifen? Dann gibt es eine Fehlermeldung. Das geht nämlich nicht.



# Strings Definieren, Verketten, und Indizieren

- Wir können einen String auch „von hinten“ indizieren. `x[-1]` liefert das letzte Zeichen des Strings `x`.



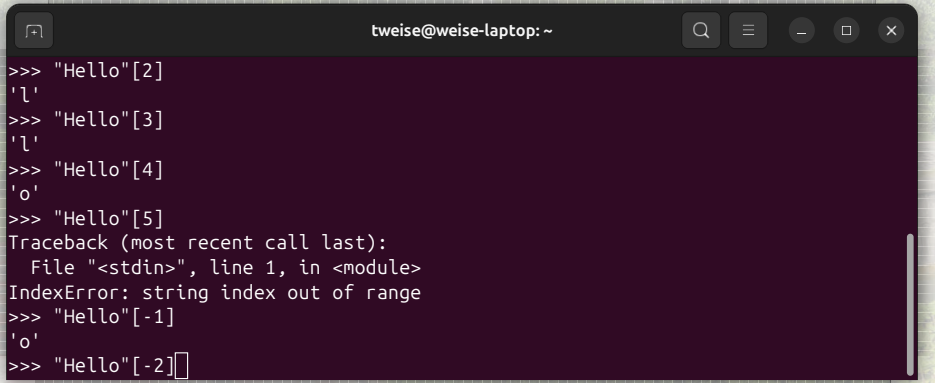
# Strings Definieren, Verketten, und Indizieren

- Wir können einen String auch „von hinten“ indizieren. `x[-1]` liefert das letzte Zeichen des Strings `x`. Bei „Hello“ ist das „o“.



# Strings Definieren, Verketten, und Indizieren

- Das vorletze Zeichen bekommen wir durch Index `[-2]`.



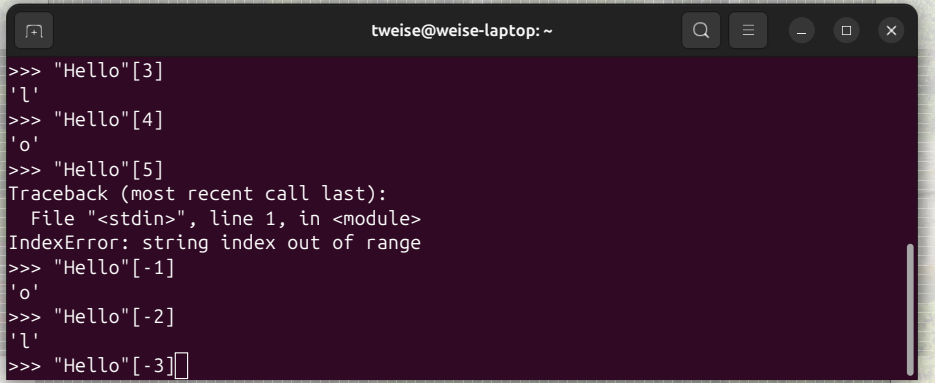
# Strings Definieren, Verketten, und Indizieren

- Das vorletze Zeichen bekommen wir durch Index `[-2]`. Bei „Hello“ ist das „l“.



# Strings Definieren, Verketten, und Indizieren

- Das vor-vorletze Zeichen bekommen wir durch Index `[-3]`.



# Strings Definieren, Verketten, und Indizieren

- Das vor-vorletze Zeichen bekommen wir durch Index `[-3]`. Bei „Hello“ ist das „l“.



# Strings Definieren, Verketten, und Indizieren



- Das vierte Zeichen von hinten bekommen wir durch Index `[-4]`.

tweise@weise-laptop: ~

```
>>> "Hello"[4]
'o'
>>> "Hello"[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> "Hello"[-1]
'o'
>>> "Hello"[-2]
'l'
>>> "Hello"[-3]
'l'
>>> "Hello"[-4]
```

# Strings Definieren, Verketten, und Indizieren

- Das vierte Zeichen von hinten bekommen wir durch Index `[-4]`. Bei „Hello“ ist das „e“.



# Strings Definieren, Verketten, und Indizieren

- Das fünfte Zeichen von hinten bekommen wir durch Index `[-5]`. Bei „Hello“ ist das „H“.



# Strings Definieren, Verketten, und Indizieren

- Das fünfte Zeichen von hinten bekommen wir durch Index `[-5]`. Bei „Hello“ ist das „H“.



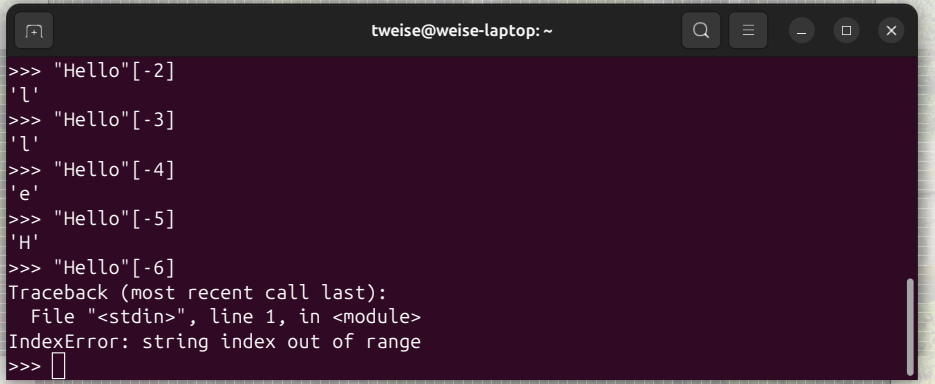
# Strings Definieren, Verketten, und Indizieren

- „Hello“ hat nur fünf Zeichen. Was passiert, wenn wir versuchen, auf das sechste Zeichen von hinten zuzugreifen?



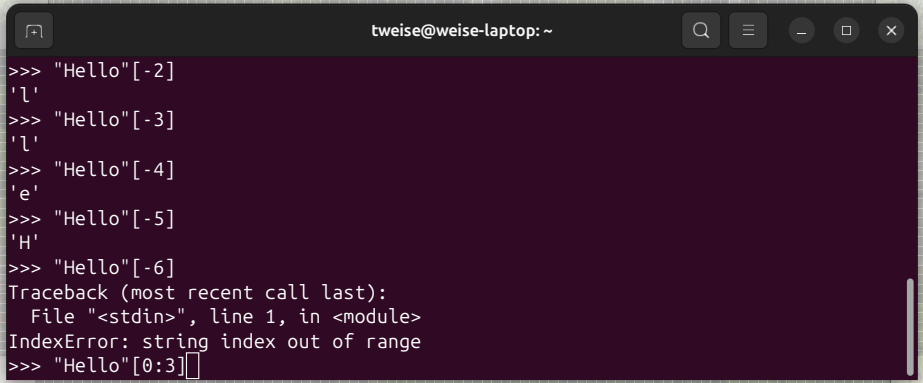
# Strings Definieren, Verketten, und Indizieren

- „Hello“ hat nur fünf Zeichen. Was passiert, wenn wir versuchen, auf das sechste Zeichen von hinten zuzugreifen? Dann gibt es wieder eine Fehlermeldung, weil das nämlich nicht geht.



# Strings Definieren, Verketten, und Indizieren

- Wir können auch ganze Substrings (Unterzeichenketten) extrahieren, in dem wir den Index `i` des ersten Zeichens und den Index `j` *nach* dem letzten zu extrahierenden Zeichen angeben als `[i:j]`.



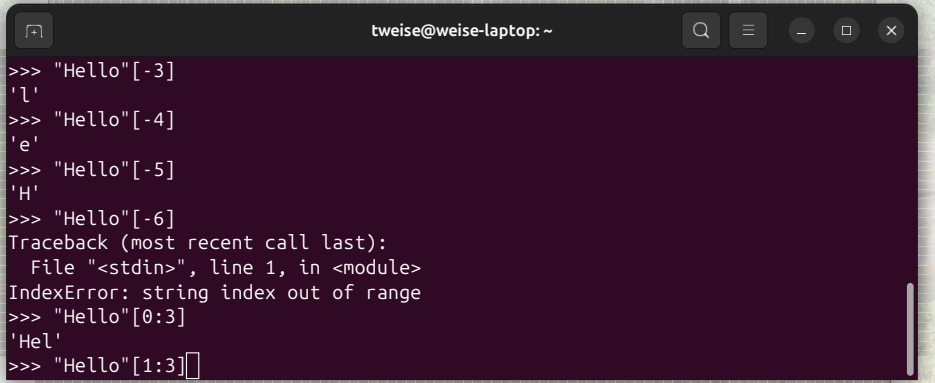
# Strings Definieren, Verketten, und Indizieren

- Wir können auch ganze Substrings (Unterzeichenketten) extrahieren, in dem wir den Index `i` des ersten Zeichens und den Index `j` *nach* dem letzten zu extrahierenden Zeichen angeben als `[i:j]`. `[0:3]` ergibt also die Zeichen an den Indizes 0, 1, 2. Bei „Hello“ ist das `"Hel"`.



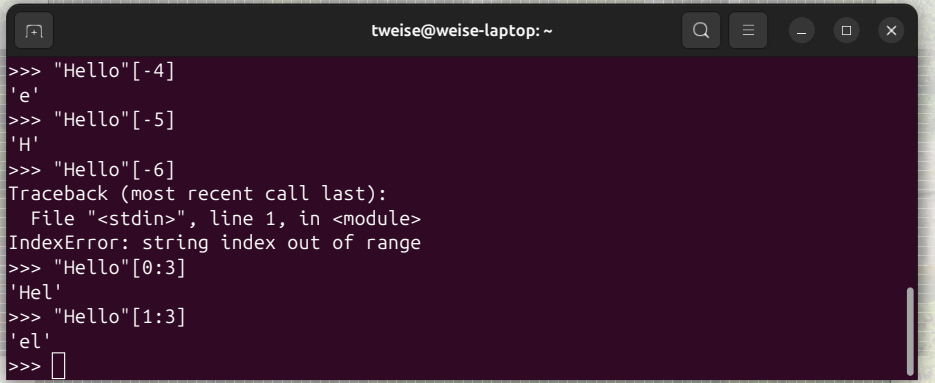
# Strings Definieren, Verketten, und Indizieren

- `"Hello"[1:3]` ist der Substring, der beim zweiten Zeichen anfängt (Index 1) und vor dem vierten Zeichen (Index 3) aufhört.



# Strings Definieren, Verketten, und Indizieren

- `"Hello"[1:3]` ist der Substring, der beim zweiten Zeichen anfängt (Index 1) und vor dem vierten Zeichen (Index 3) aufhört. Also `"el"`.



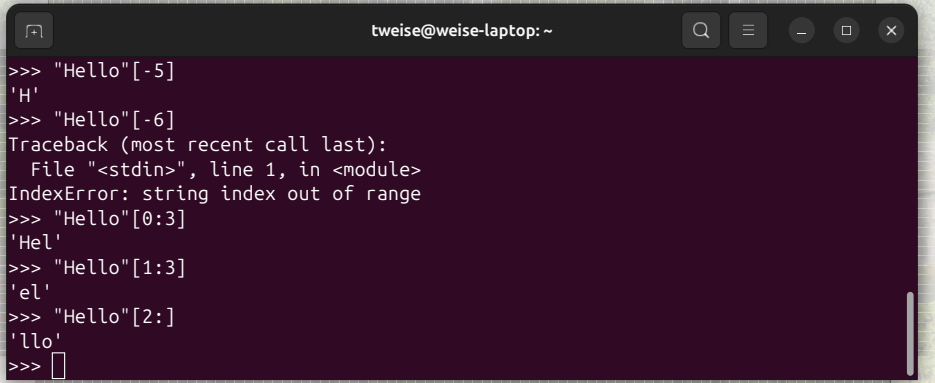
# Strings Definieren, Verketten, und Indizieren

- Lassen wir einfach den zweiten Index weg, dann werden alle Zeichen bis zum Ende des Strings zurückgegeben.



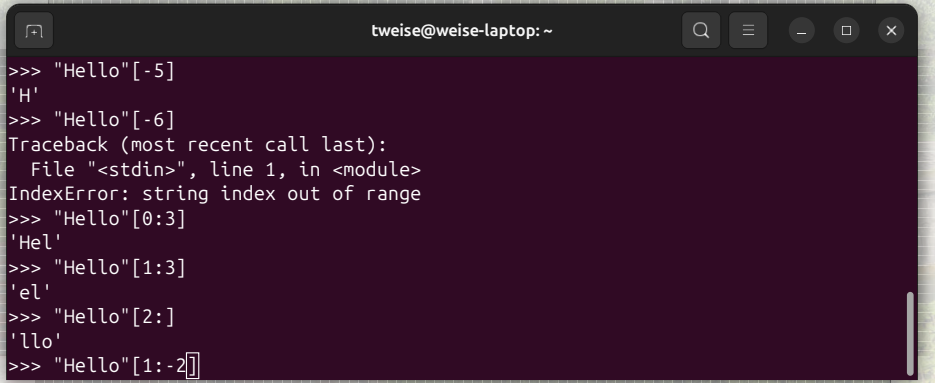
# Strings Definieren, Verketten, und Indizieren

- Lassen wir einfach den zweiten Index weg, dann werden alle Zeichen bis zum Ende des Strings zurückgegeben. `"Hello"[2:]` ergibt also die Zeichen an den Indices 2, 3, 4, und 5, also `"llo"`.



# Strings Definieren, Verketten, und Indizieren

- Wir können genausogut auch negative Indizes verwenden, die dann wieder von hinten Zählen.



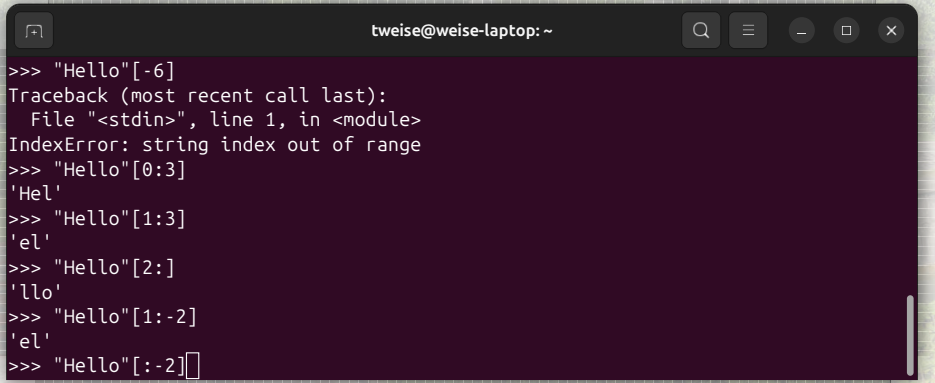
# Strings Definieren, Verketten, und Indizieren

- Wir können genausogut auch negative Indizes verwenden, die dann wieder von hinten Zählen. `"Hello"[1:-2]` started an Index 1 und hört vor dem zweiten Zeichen von hinten (Index 3) auf. Die Indizes im Originalstring gehen von 0 bis 5, wir haben also die Zeichen an den Indizes von 1 bis 2, demach also `"el"`.



# Strings Definieren, Verketten, und Indizieren

- Wir können auch den ersten Index weglassen, dann fängt der zurückgelieferte String am Anfang der Zeichenkette an und geht bis vor den zweiten Index.



# Strings Definieren, Verketten, und Indizieren

- Wir können auch den ersten Index weglassen, dann fängt der zurückgelieferte String am Anfang der Zeichenkette an und geht bis vor den zweiten Index. `"Hello"[:-2]` fängt am Anfang an und hört vor dem vorletzten Zeichen auf, ist also `"Hel"`.



# Strings Definieren, Verketten, und Indizieren



- Wir können auch drei Werte angeben, `[i:j:k]`, wobei `i` wieder der Index des ersten Zeichens und `j` der Index nach dem letzten Zeichen ist. `k` ist die Schrittweite. `"Hello World!"[1:8:2]` liefert jedes zweite Zeichen beginnend an Index 1 und endent vor Index 8, also die Zeichen an den Indices 1, 3, 5, und 7. Das sind `"el o"`.



# Strings Definieren, Verketten, und Indizieren

- Wir können auch drei Werte angeben, `[i:j:k]`, wobei `i` wieder der Index des ersten Zeichens und `j` der Index nach dem letzten Zeichen ist. `k` ist die Schrittweite. `"Hello World!"[1:8:2]` liefert jedes zweite Zeichen beginnend an Index 1 und endent vor Index 8, also die Zeichen an den Indices 1, 3, 5, und 7. Das sind `"el o"`.



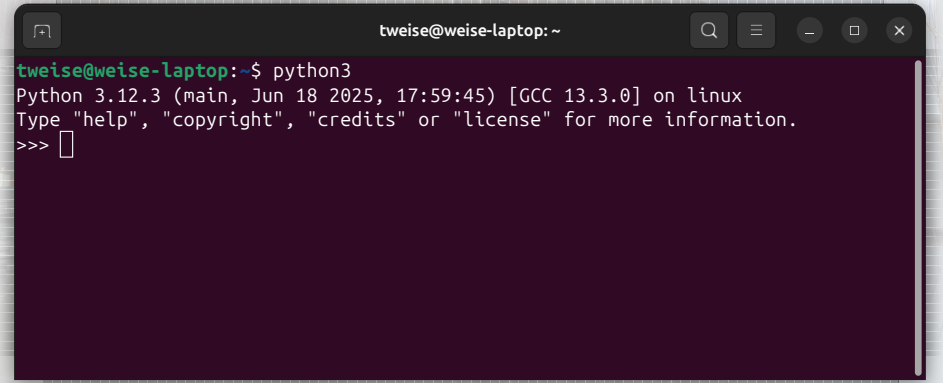
## Strings Definieren, Verketten, und Indizieren



- Schauen wir uns die grundlegenden Operationen für Strings an.
- Damit haben wir also erstmal ein Grundverständnis, was Strings sind, wie wir sie verketten, ihre Länge bestimmen, und wieder auseinanderfummeln können.

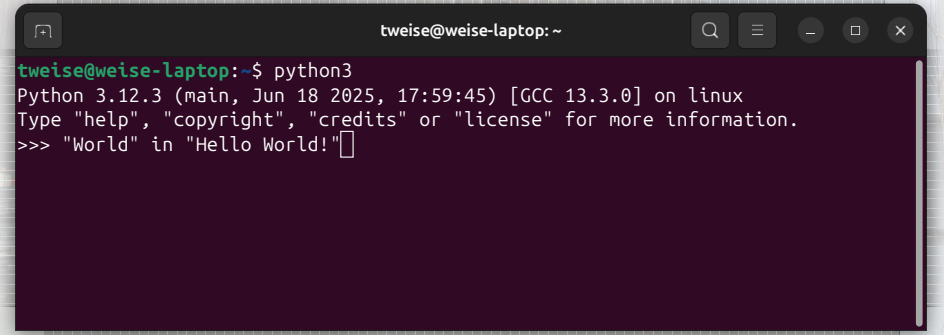
# Weitere Grundlegende String Operationen

- Schauen wir uns die ein paar weitere grundlegenden Operationen für Strings an.



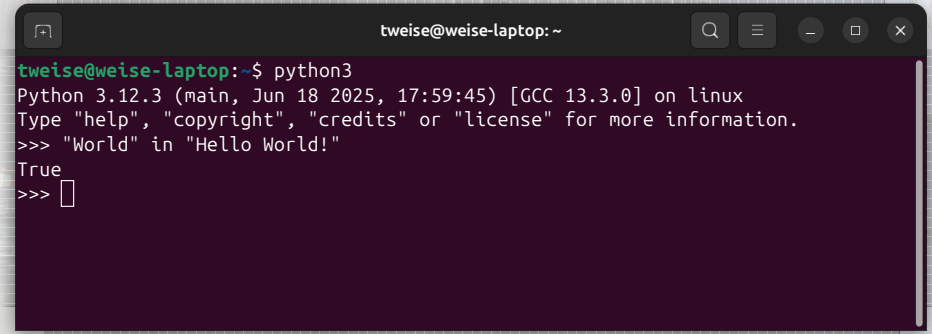
# Weitere Grundlegende String Operationen

- Mit dem Operator `a in b` prüfen wir, ob die Zeichenkette `a` irgendwo im String `b` enthalten ist.



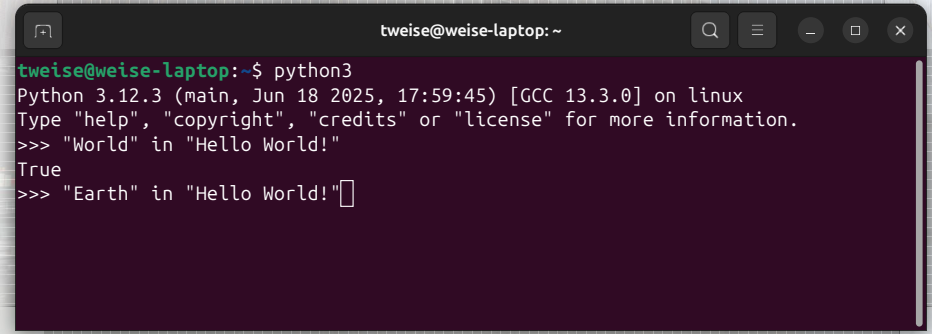
# Weitere Grundlegende String Operationen

- Mit dem Operator `a in b` prüfen wir, ob die Zeichenkette `a` irgendwo im String `b` enthalten ist. Das ist hier der Fall: `"World"` ist tatsächlich im String `"Hello World!"` enthalten.



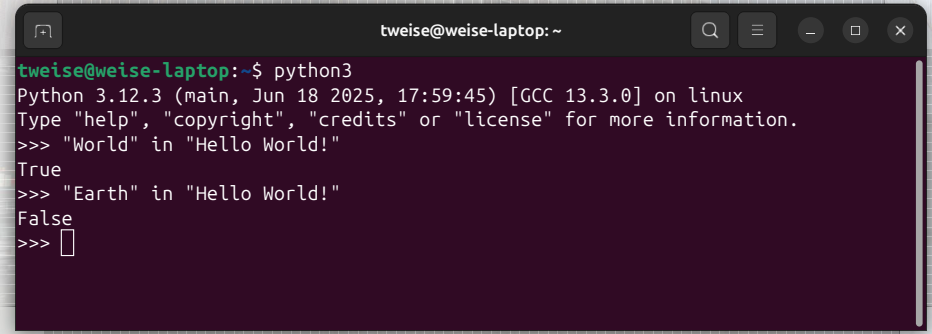
# Weitere Grundlegende String Operationen

- Ist `"Earth"` irgendwo in `"Hello World!"` enthalten?



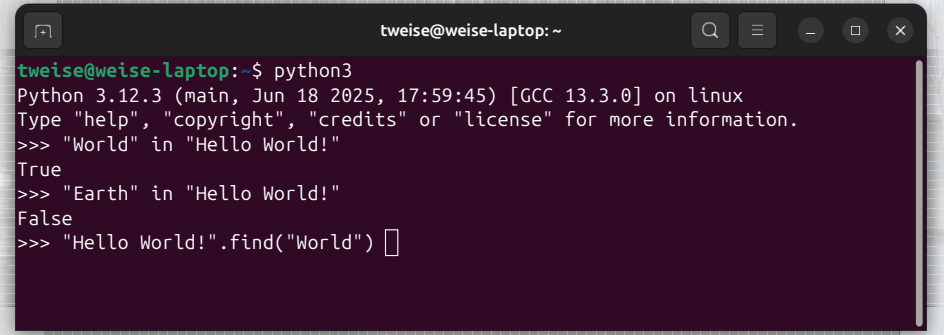
# Weitere Grundlegende String Operationen

- Ist `"Earth"` irgendwo in `"Hello World!"` enthalten? Nein.



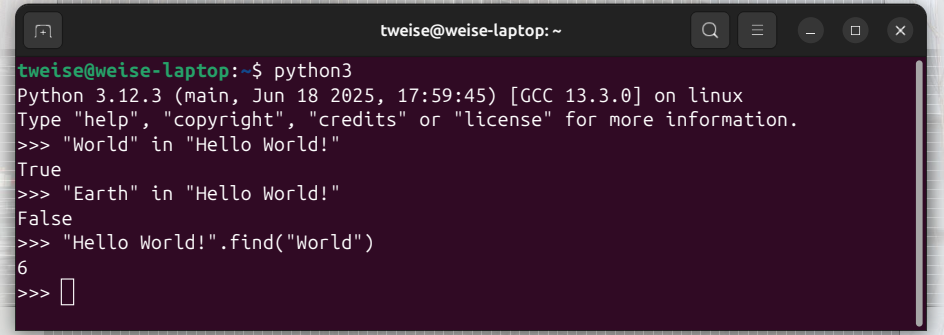
# Weitere Grundlegende String Operationen

- Die Funktion `a.find(b)` sucht den Index, an dem die Zeichenkette `b` in `a` beginnt.



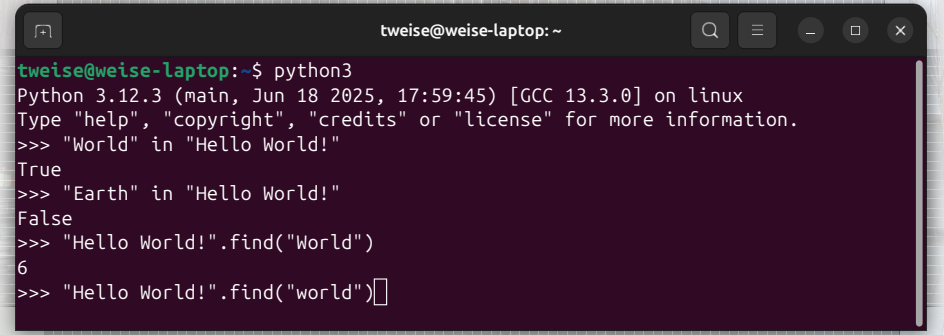
# Weitere Grundlegende String Operationen

- Die Funktion `a.find(b)` sucht den Index, an dem die Zeichenkette `b` in `a` beginnt. `"World"` beginnt an Index 6 in `"Hello World!"`.



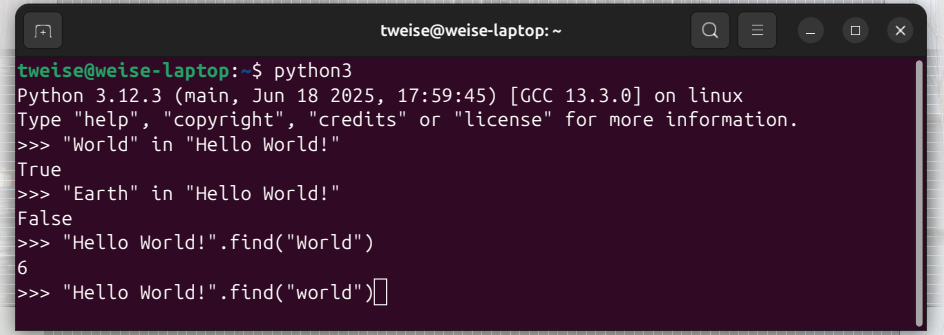
# Weitere Grundlegende String Operationen

- String-Funktionen und Vergleiche sind schreibungsabhängig (EN: *case-sensitive*): Groß- und Kleinbuchstaben werden als unterschiedlich betrachtet.



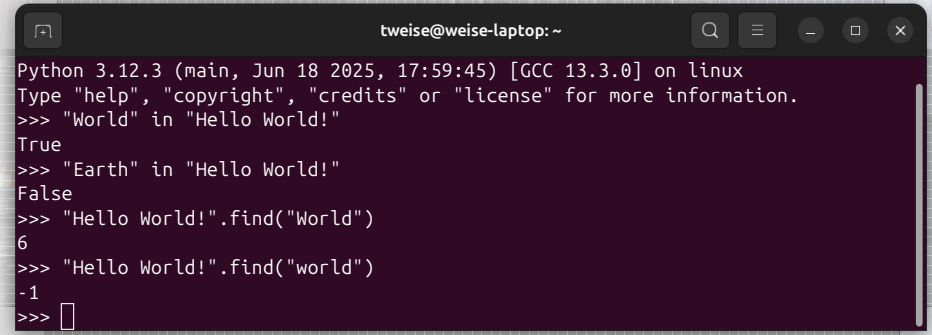
# Weitere Grundlegende String Operationen

- String-Funktionen und Vergleiche sind schreibungsabhängig (EN: *case-sensitive*): Groß- und Kleinbuchstaben werden als unterschiedlich betrachtet. Somit gilt `"W" != "w"`.



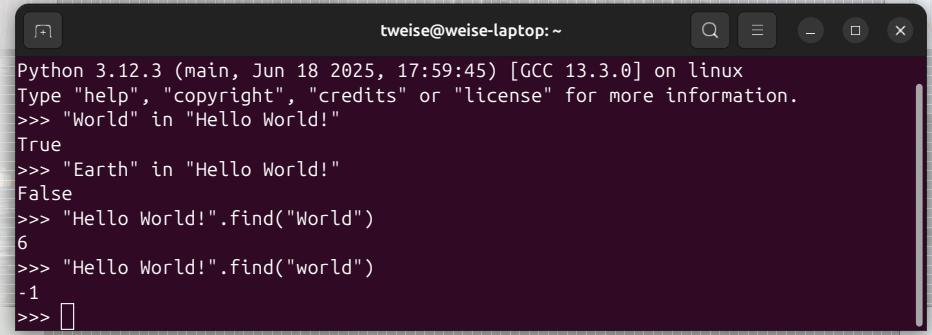
# Weitere Grundlegende String Operationen

- String-Funktionen und Vergleiche sind schreibungsabhängig (EN: *case-sensitive*): Groß- und Kleinbuchstaben werden als unterschiedlich betrachtet. Somit gilt `"W" != "w"`. Somit kann `"world"` nicht in `"Hello World!"` gefunden werden.



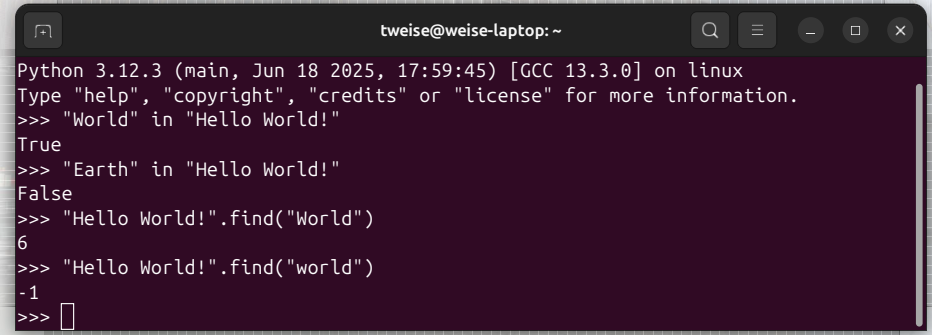
# Weitere Grundlegende String Operationen

- Groß- und Kleinbuchstaben werden als unterschiedlich betrachtet. Somit gilt `"W" != "w"`. Somit kann `"world"` nicht in `"Hello World!"` gefunden werden. Somit liefert die Funktion `-1` zurück.



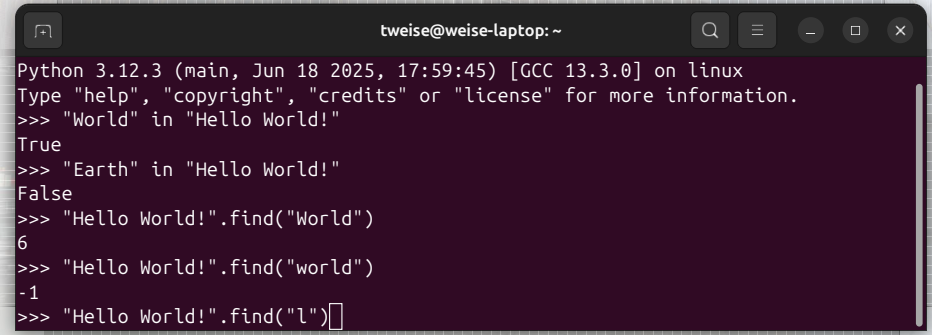
# Weitere Grundlegende String Operationen

- Groß- und Kleinbuchstaben werden als unterschiedlich betrachtet. Somit kann `"world"` nicht in `"Hello World!"` gefunden werden. Somit liefert die Funktion `-1` zurück. Beachte also: Niemals das Ergebnis von `find` direkt zum Indizieren nehmen, denn `-1` steht für „letztes Zeichen“...



# Weitere Grundlegende String Operationen

- Wo befindet sich `"l"` in `"Hello World!"`?



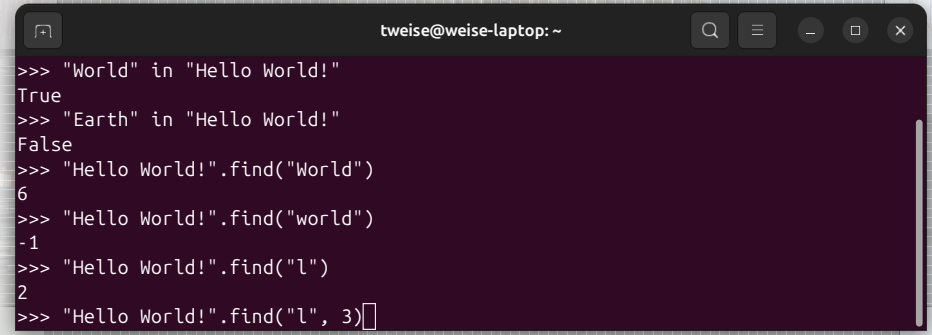
# Weitere Grundlegende String Operationen

- Wo befindet sich `"l"` in `"Hello World!"`? An Index 2.



# Weitere Grundlegende String Operationen

- Wir können auch den Index angeben, ab dem gesucht werden soll: Wo befindet sich `"l"` in `"Hello World!"` *wenn wir ab Index 3 suchen*?



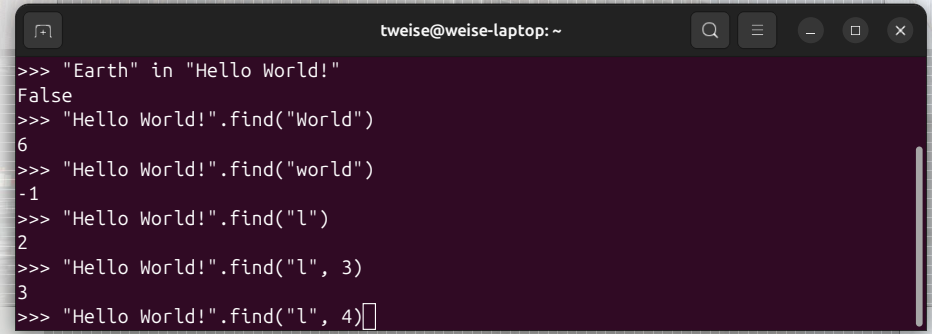
# Weitere Grundlegende String Operationen

- Wir können auch den Index angeben, ab dem gesucht werden soll: Wo befindet sich `"l"` in `"Hello World!"` *wenn wir ab Index 3 suchen*? An Index 3.



# Weitere Grundlegende String Operationen

- Wo befindet sich `"l"` in `"Hello World!"` wenn wir ab Index 4 suchen?



# Weitere Grundlegende String Operationen

- Wo befindet sich `"l"` in `"Hello World!"` wenn wir ab Index 4 suchen? An Index 9.



# Weitere Grundlegende String Operationen

- Wo befindet sich `"l"` in `"Hello World!"` wenn wir ab Index 10 suchen?



# Weitere Grundlegende String Operationen

- Wo befindet sich `"l"` in `"Hello World!"` wenn wir ab Index 10 suchen? Dann finden wir kein „l“ mehr und `-1` wird zurückgegeben.



# Weitere Grundlegende String Operationen

- `rfind` sucht von hinten/rechts nach vorne.



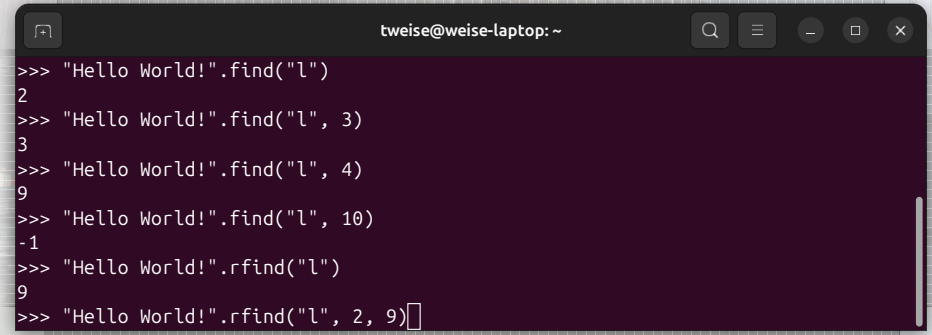
# Weitere Grundlegende String Operationen

- `rfind` sucht von hinten/rechts nach vorne. Von rechts aus gehen findet sich das erste Auftreten von `"l"` in `"Hello World!"` an Index 9.



# Weitere Grundlegende String Operationen

- Sowohl bei `find` als auch bei `rfind` können wir keinen Index, den Startindex, oder den Start- und den (exklusiven) End-Index für die Suche angeben. Wir suchen nun in der Zeichekette von Index 2 bis vor Index 9 von rechts.



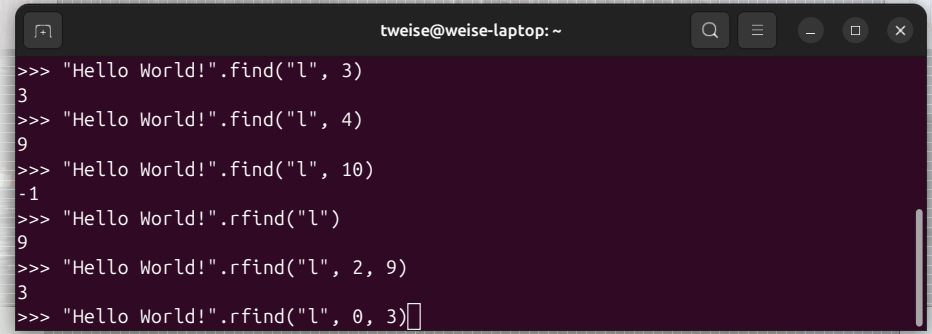
# Weitere Grundlegende String Operationen

- Sowohl bei `find` als auch bei `rfind` können wir keinen Index, den Startindex, oder den Start- und den (exklusiven) End-Index für die Suche angeben. Wir suchen nun in der Zeichekette von Index 2 bis vor Index 9 von rechts. Und finden `"l"` an Index 3.



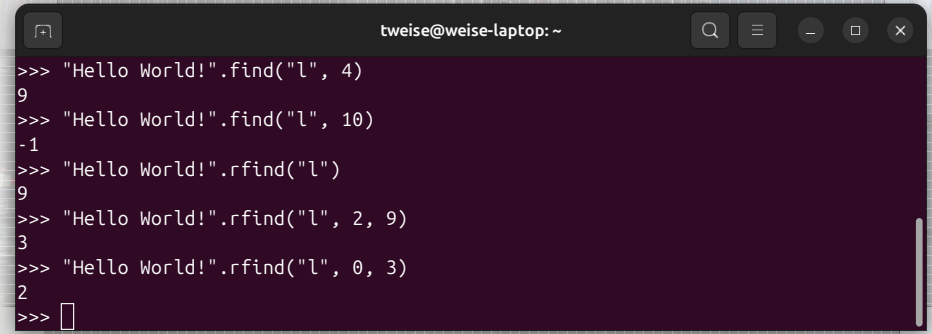
# Weitere Grundlegende String Operationen

- Wir suchen nun in der Zeichekette von Index 0 bis vor Index 3 von rechts.



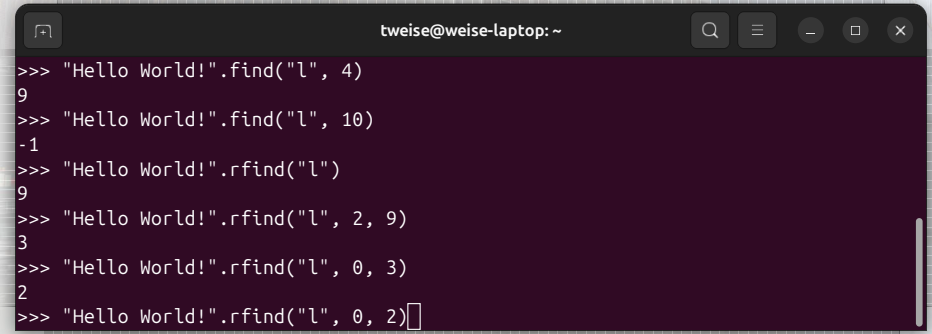
# Weitere Grundlegende String Operationen

- Wir suchen nun in der Zeichekette von Index 0 bis vor Index 3 von rechts. Und finden `"l"` an Index 2.



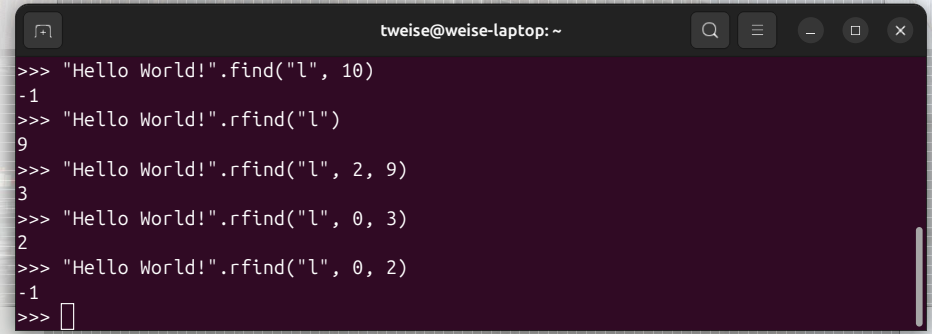
# Weitere Grundlegende String Operationen

- Wir suchen nun in der Zeichekette von Index 0 bis vor Index 2 von rechts.



# Weitere Grundlegende String Operationen

- Wir suchen nun in der Zeichekette von Index 0 bis vor Index 2 von rechts. Und finden `"l"` gar nicht mehr, bekommen also `-1` zurück.



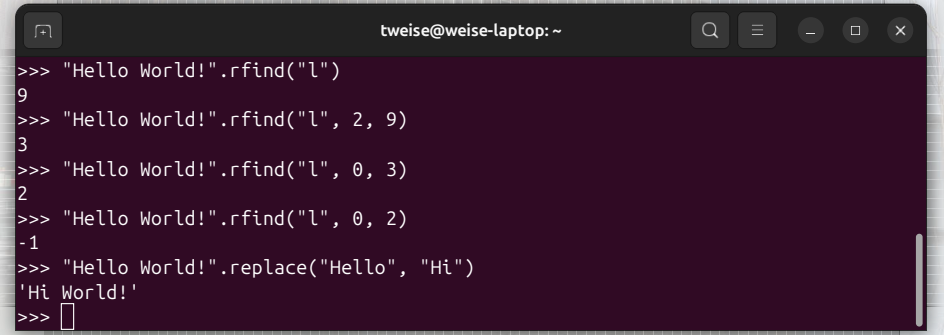
# Weitere Grundlegende String Operationen

- `a.replace(b, c)` ersetzt alle Auftreten von `b` in `a` mit `c` und gibt das Ergebnis als neuen String zurück



# Weitere Grundlegende String Operationen

- `a.replace(b, c)` ersetzt alle Auftreten von `b` in `a` mit `c` und gibt das Ergebnis als neuen String zurück Ersetzen wir alle `"Hello"` in `"Hello World!"` mit `"Hi"`, so bekommen wir `"Hi World!"`.



# Weitere Grundlegende String Operationen

- Ersetzen wir alle `"Hello"` in `"Hello World! Hello!"` mit `"Hi"`...



# Weitere Grundlegende String Operationen

- Ersetzen wir alle `"Hello"` in `"Hello World! Hello!"` mit `"Hi"`, so bekommen wir `"Hi World! Hi!"`.



# Weitere Grundlegende String Operationen

- Ersetzen wir alle `"Hello"` in `"Hello World!"` mit `"Hello! Hello!"`...



# Weitere Grundlegende String Operationen

- Ersetzen wir alle `"Hello"` in `"Hello World!"` mit `"Hello! Hello!"`, so bekommen wir `"Hello! Hello! World!"`. Das Ersetzen funktioniert also nicht rekursiv, ersetzt also nicht in bereits ersetzten Strings.



# Weitere Grundlegende String Operationen

- `a.strip()` entfernt alle so-genannten „whitespace“-Zeichen (Leerzeichen, Newlines, Tabs) am Anfang und Ende eines Strings und gibt das Ergebnis als neuen String zurück



# Weitere Grundlegende String Operationen

- `a.strip()` entfernt alle so-genannten „whitespace“-Zeichen (Leerzeichen, Newlines, Tabs) am Anfang und Ende eines Strings und gibt das Ergebnis als neuen String zurück
  `" Hello World! "` wird so zu `"Hello World!"`.



# Weitere Grundlegende String Operationen

- `a.lstrip()` entfernt alle so-genannten „whitespace“-Zeichen (Leerzeichen, Newlines, Tabs) am Anfang eines Strings und gibt das Ergebnis als neuen String zurück



# Weitere Grundlegende String Operationen

- `a.lstrip()` entfernt alle so-genannten „whitespace“-Zeichen (Leerzeichen, Newlines, Tabs) am Anfang eines Strings und gibt das Ergebnis als neuen String zurück
  `" Hello World! "` wird so zu `"Hello World! "`.



# Weitere Grundlegende String Operationen

- `a.rstrip()` entfernt alle so-genannten „whitespace“-Zeichen (Leerzeichen, Newlines, Tabs) am Ende eines Strings und gibt das Ergebnis als neuen String zurück



# Weitere Grundlegende String Operationen

- `a.rstrip()` entfernt alle so-genannten „whitespace“-Zeichen (Leerzeichen, Newlines, Tabs) am Ende eines Strings und gibt das Ergebnis als neuen String zurück
  `" Hello World! "` wird so zu `" Hello World!"`.



# Weitere Grundlegende String Operationen

- `a.lower()` wandelt alle Großbuchstaben in `a` in Kleinbuchstaben um und gibt das Ergebnis als neuen String zurück.



# Weitere Grundlegende String Operationen

- `a.lower()` wandelt alle Großbuchstaben in `a` in Kleinbuchstaben um und gibt das Ergebnis als neuen String zurück. `"Hello World!"` wird so zu `"hello world!"`.



# Weitere Grundlegende String Operationen

- `a.upper()` wandelt alle Kleinbuchstaben in `a` in Großbuchstaben um und gibt das Ergebnis als neuen String zurück.



# Weitere Grundlegende String Operationen

- `a.upper()` wandelt alle Kleinbuchstaben in `a` in Großbuchstaben um und gibt das Ergebnis als neuen String zurück. `"Hello World!"` wird so zu `"HELLOW WORLD"`.



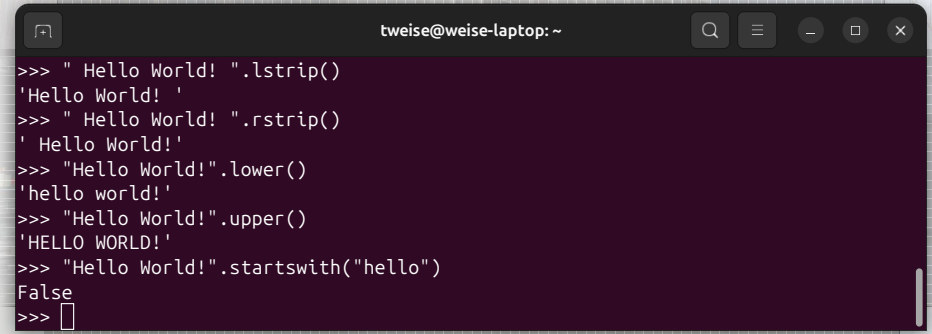
# Weitere Grundlegende String Operationen

- `a.startswith(b)` gibt `True` zurück wenn und nur wenn `a` mit `b` anfängt.



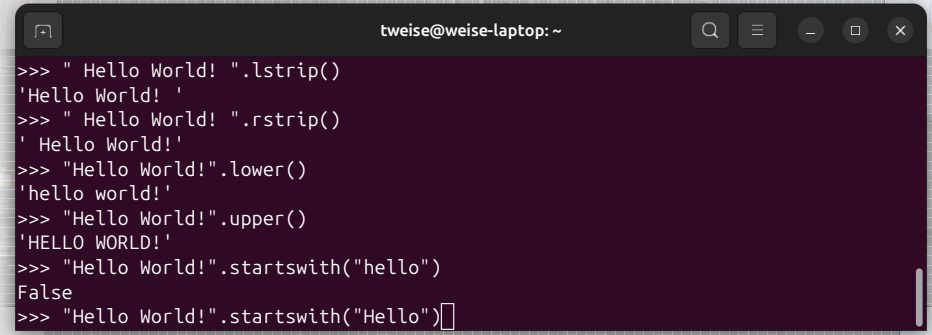
# Weitere Grundlegende String Operationen

- `a.startswith(b)` gibt `True` zurück wenn und nur wenn `a` mit `b` anfängt. Und weil es case-sensitive ist, ergibt das hier `False`.



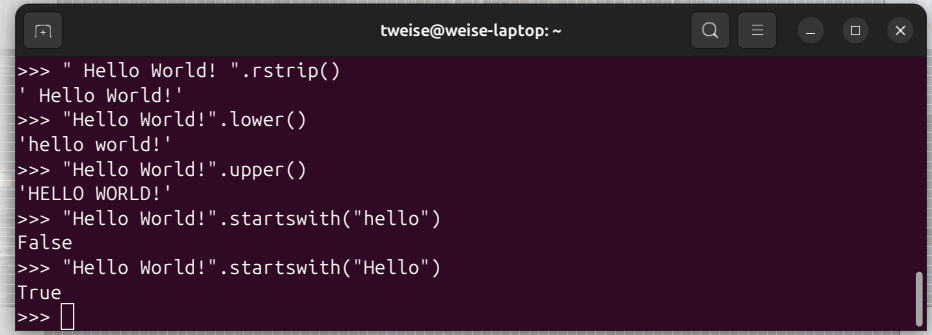
# Weitere Grundlegende String Operationen

- `a.startswith(b)` gibt `True` zurück wenn und nur wenn `a` mit `b` anfängt.



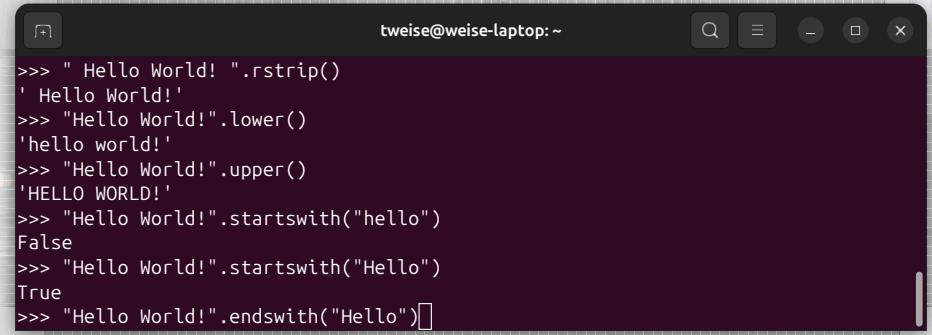
# Weitere Grundlegende String Operationen

- `a.startswith(b)` gibt `True` zurück wenn und nur wenn `a` mit `b` anfängt. Aber jetzt stimmt es.



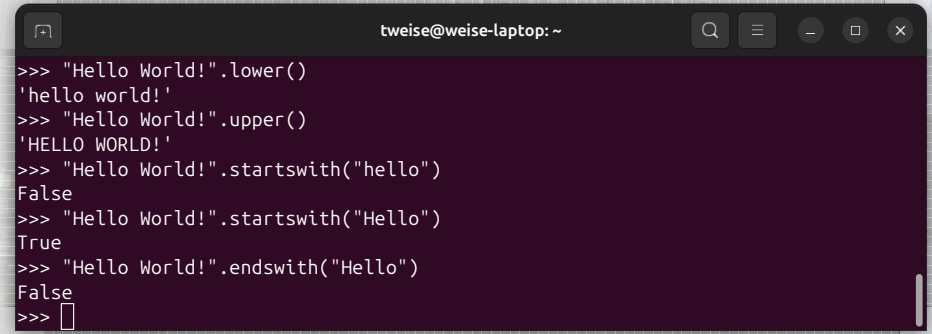
# Weitere Grundlegende String Operationen

- `a.endswith(b)` gibt `True` zurück wenn und nur wenn `a` mit `b` endet.



# Weitere Grundlegende String Operationen

- `a.endswith(b)` gibt `True` zurück wenn und nur wenn `a` mit `b` endet. Und das ist hier natürlich nicht der Fall.



# Weitere Grundlegende String Operationen

- `a.endswith(b)` gibt `True` zurück wenn und nur wenn `a` mit `b` endet.



# Weitere Grundlegende String Operationen

- `a.endswith(b)` gibt `True` zurück wenn und nur wenn `a` mit `b` endet. Aber das stimmt.



## Weitere Grundlegende String Operationen

- Schauen wir uns die ein paar weitere grundlegenden Operationen für Strings an.
- So, nun haben wir schon ziemlich viele String-Funktionen gelernt.

# Die Funktion str und f-Strings



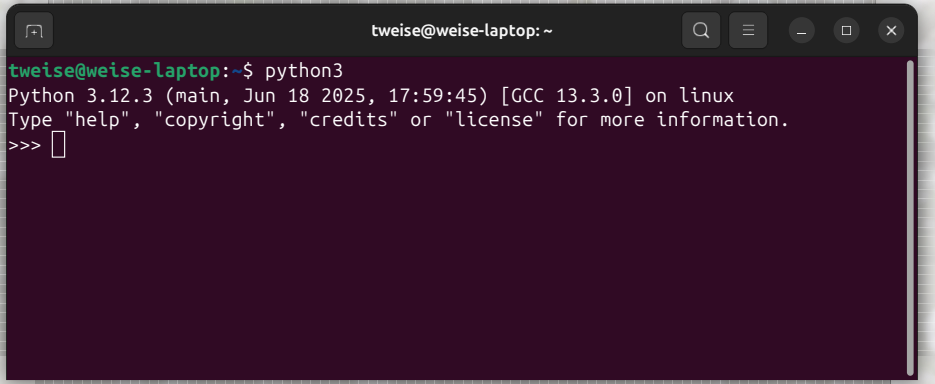
# Die Funktion str



- Für Werte `a` der meisten Datentypen liefert die Funktion `str(a)` einen entsprechenden String zurück.

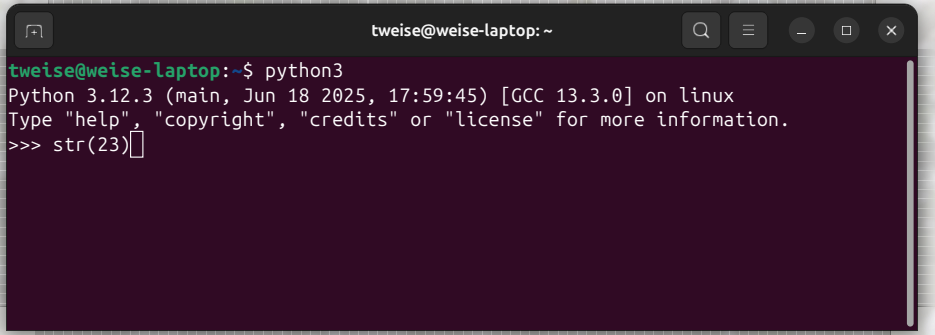
# Die Funktion str

- Für Werte `a` der meisten Datentypen liefert die Funktion `str(a)` einen entsprechenden String zurück.
- Probieren wir das mal aus.



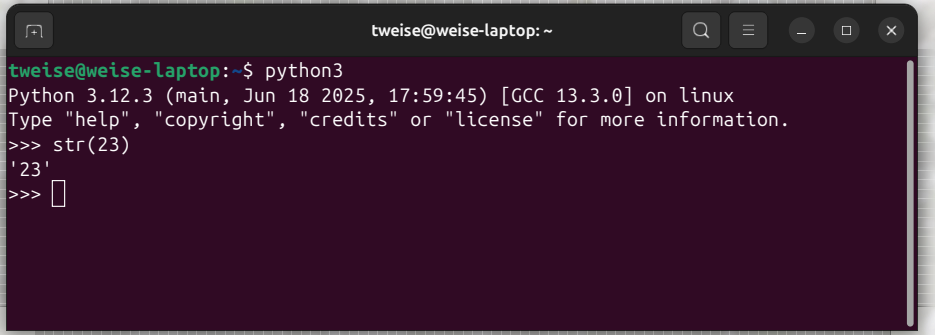
# Die Funktion str

- Wenden wir die Funktion `str` auf den `int`-Wert `23` an...



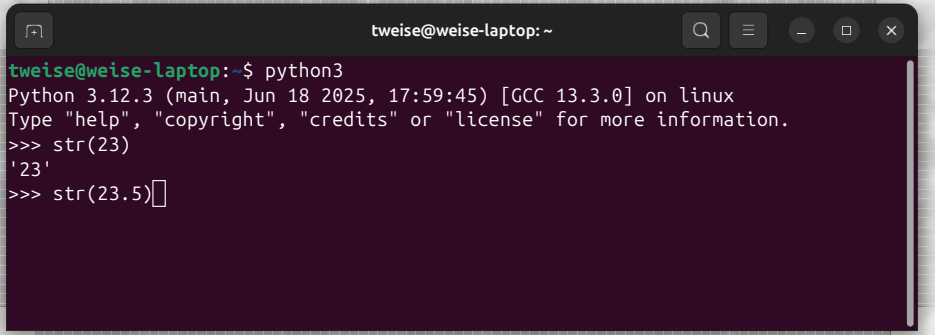
# Die Funktion str

- Wenden wir die Funktion `str` auf den `int`-Wert `23` an so liefert sie den `str`-Wert `"23"`.



# Die Funktion str

- Wenden wir die Funktion `str` auf den `float`-Wert `23.5` an...



# Die Funktion str

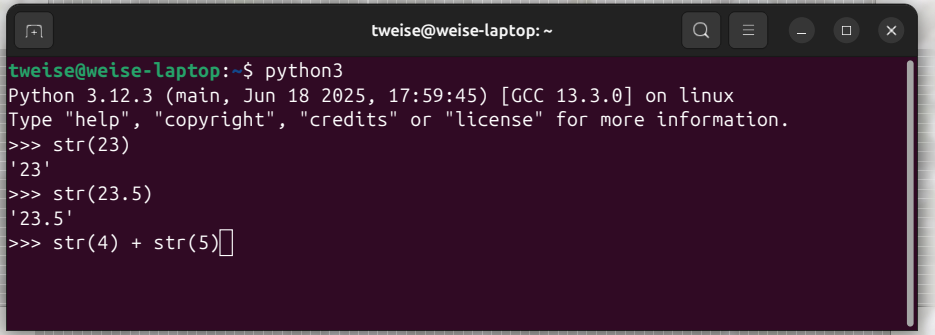


- Wenden wir die Funktion `str` auf den `float`-Wert `23.5` an so liefert sie den `str`-Wert `"23.5"`.

```
tweise@weise-laptop: ~
tweise@weise-laptop:~$ python3
Python 3.12.3 (main, Jun 18 2025, 17:59:45) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> str(23)
'23'
>>> str(23.5)
'23.5'
>>>
```

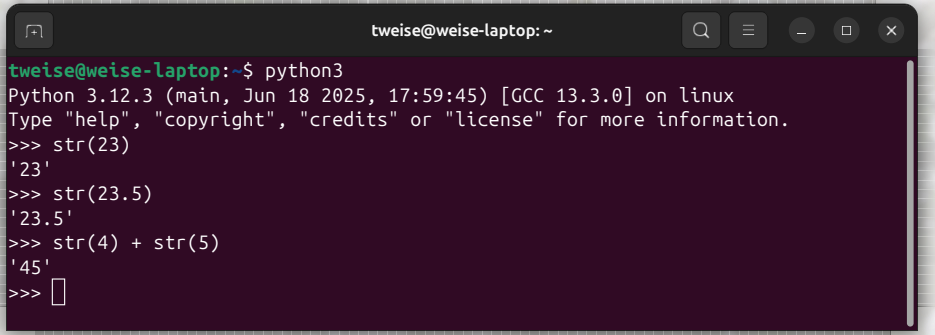
# Die Funktion str

- Wenn der Ausdruck `str(4) + str(5)` ausgerechnet wird...



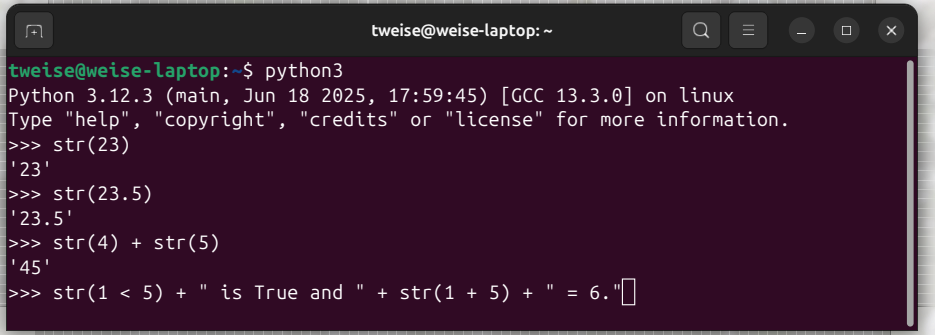
# Die Funktion str

- Wenn der Ausdruck `str(4) + str(5)` ausgerechnet wird, dann werden zuerst die beiden `str`-Funktionen ausgerechnet, was dann `"4" + "5"` ergibt. Dies ergibt dann wiederum `"45"`.



# Die Funktion str

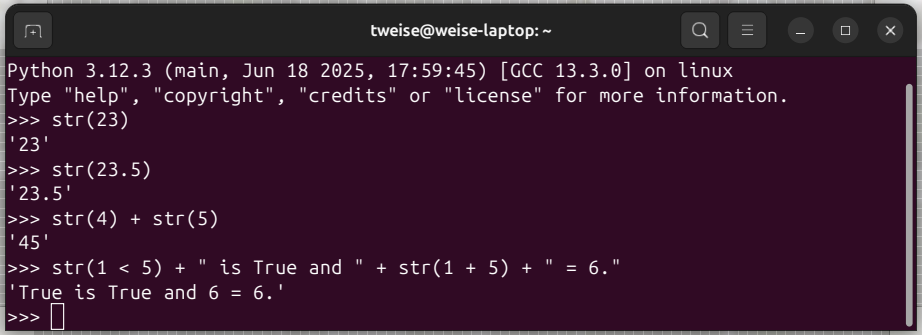
- Wir können nahezu beliebige Aufrufe von `str(...)` mit anderen Strings und `+` kombinieren.



# Die Funktion str



- Wir können nahezu beliebige Aufrufe von `str(...)` mit anderen Strings und `+` kombinieren. Beispielsweise ist `str(1 < 5)` das selbe wie `str(True)`, was `"True"` ergibt. `str(1 + 5)` ist das selbe wie `str(6)`, was dann `"6"` ergibt. Alle Strings zusammen ergeben dann...



# f-strings



- `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."` ... das sieht ziemlich umständlich aus.

## f-strings



- `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."` ... das sieht ziemlich umständlich aus.
- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können.

# f-strings



- `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."` ... das sieht ziemlich umständlich aus.
- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können: f-Strings.

# f-strings



- `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."` ... das sieht ziemlich umständlich aus.
- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können: f-Strings.
- f-Strings erlauben es uns, die Ergebnisse von Ausdrücken in Zeichenketten einzufügen[6,25,26,28,42,53].

# f-strings



- `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."` ... das sieht ziemlich umständlich aus.
- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können: f-Strings.
- f-Strings erlauben es uns, die Ergebnisse von Ausdrücken in Zeichenketten einzufügen[6,25,26,28,42,53].
- Sie haben eine einfache Syntax.

# f-strings

- `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."` ... das sieht ziemlich umständlich aus.
- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können: f-Strings.
- f-Strings erlauben es uns, die Ergebnisse von Ausdrücken in Zeichenketten einzufügen[6,25,26,28,42,53].
- Sie haben eine einfache Syntax:
  1. Sie sind mit `f"..."` begrenzt (beachten Sie das kleine `f` vor dem einleitenden Anführungszeichen!).

# f-strings



- `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."` ... das sieht ziemlich umständlich aus.
- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können: f-Strings.
- f-Strings erlauben es uns, die Ergebnisse von Ausdrücken in Zeichenketten einzufügen[6,25,26,28,42,53].
- Sie haben eine einfache Syntax:
  1. Sie sind mit `f"..."` begrenzt (beachten Sie das kleine `f` vor dem einleitenden Anführungszeichen!).
  2. Sie können beliebige Ausdrücke enthalten, die jeweils mit `{...}` begrenzt sind.

# f-strings



- `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."` ... das sieht ziemlich umständlich aus.
- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können: f-Strings.
- f-Strings erlauben es uns, die Ergebnisse von Ausdrücken in Zeichenketten einzufügen[6,25,26,28,42,53].
- Sie haben eine einfache Syntax:
  1. Sie sind mit `f"..."` begrenzt (beachten Sie das kleine `f` vor dem einleitenden Anführungszeichen!).
  2. Sie können beliebige Ausdrücke enthalten, die jeweils mit `{...}` begrenzt sind.
  3. Zusätzlich können Markierungen für die Formatierung der Ausdruck-Ergebnisse angegeben werden.

# f-strings



- `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."` ... das sieht ziemlich umständlich aus.
- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können: f-Strings.
- f-Strings erlauben es uns, die Ergebnisse von Ausdrücken in Zeichenketten einzufügen[6,25,26,28,42,53].
- Sie haben eine einfache Syntax:
  1. Sie sind mit `f"..."` begrenzt (beachten Sie das kleine `f` vor dem einleitenden Anführungszeichen!).
  2. Sie können beliebige Ausdrücke enthalten, die jeweils mit `{...}` begrenzt sind.
  3. Zusätzlich können Markierungen für die Formatierung der Ausdruck-Ergebnisse angegeben werden.
  4. Während der sogenannten String-Interpolation, welche automatisch passiert, werden diese Ausdrücke dann ausgerechnet und ihre Ergebnisse an ihrer Stelle in den Text eingefügt.

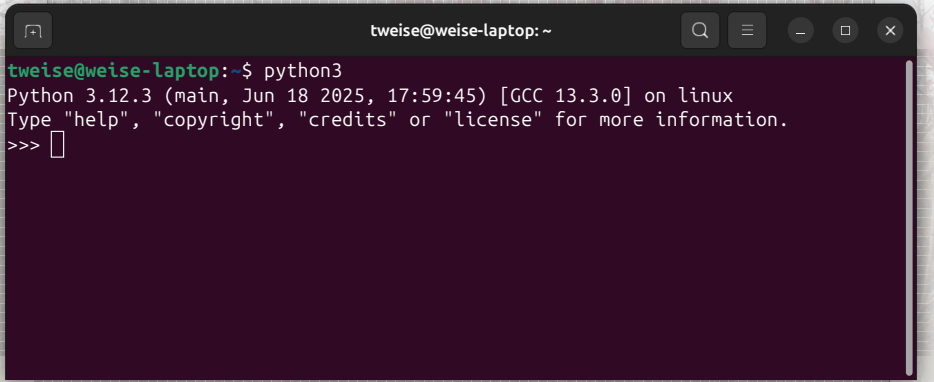
# f-strings



- `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."` ... das sieht ziemlich umständlich aus.
- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können: f-Strings.
- f-Strings erlauben es uns, die Ergebnisse von Ausdrücken in Zeichenketten einzufügen[6,25,26,28,42,53].
- Sie haben eine einfache Syntax:
  1. Sie sind mit `f"..."` begrenzt (beachten Sie das kleine `f` vor dem einleitenden Anführungszeichen!).
  2. Sie können beliebige Ausdrücke enthalten, die jeweils mit `{...}` begrenzt sind.
  3. Zusätzlich können Markierungen für die Formatierung der Ausdruck-Ergebnisse angegeben werden.
  4. Während der sogenannten String-Interpolation, welche automatisch passiert, werden diese Ausdrücke dann ausgerechnet und ihre Ergebnisse an ihrer Stelle in den Text eingefügt.
- So wird aus `f"a{6-1}b"` dann `"a5b"`.

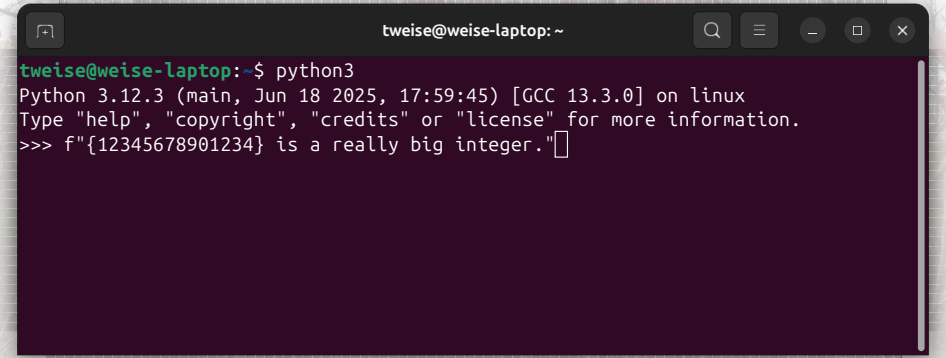
# f-strings

- Probieren wir das mal aus.



# f-strings

- Eine `int`-Zahl ist ein Ausdruck. Schreiben wir also mal einen f-String, der eine Zahl in einen Text umwandelt und mit anderem Text kombiniert.



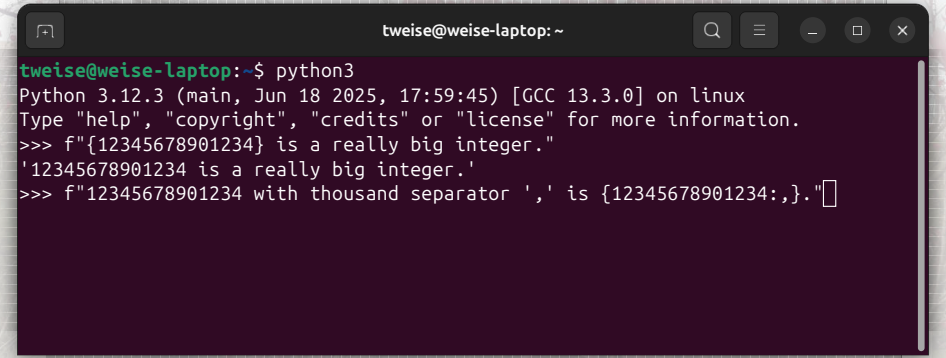
# f-strings

- Eine `int`-Zahl ist ein Ausdruck. Schreiben wir also mal einen f-String, der eine Zahl in einen Text umwandelt und mit anderem Text kombiniert. Das funktioniert problemlos.



# f-strings

- f-Strings erlauben es uns auch, die Erbenisse von Ausdrücken zu formatieren. Schreiben wir z. B. `:,` hinten an den Ausdruck (hier: der Zahl) ran, dann wird der Ausdruck als Zahl behandelt und es werden Tausender-Trennzeichen (hier: Kommas) bei der Umwandlung in Text eingefügt.



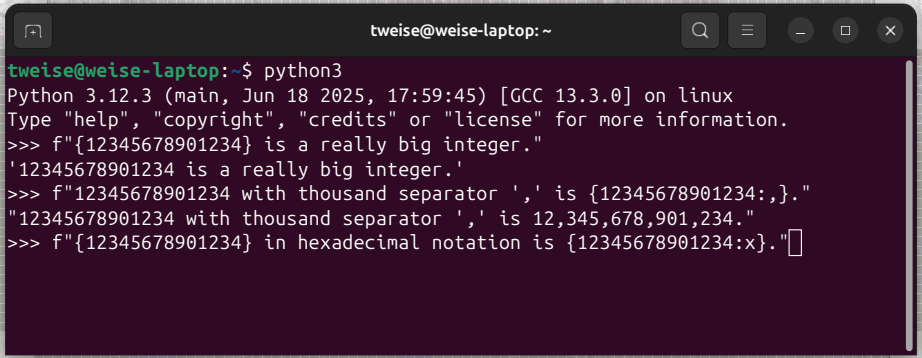
# f-strings

- f-Strings erlauben es uns auch, die Erbenisse von Ausdrücken zu formatieren. Schreiben wir z. B. `:,` hinten an den Ausdruck (hier: der Zahl) ran, dann wird der Ausdruck als Zahl behandelt und es werden Tausender-Trennzeichen (hier: Kommas) bei der Umwandlung in Text eingefügt. Aus der Zahl `12345678901234` wird der Text `12,345,678,901,234`.



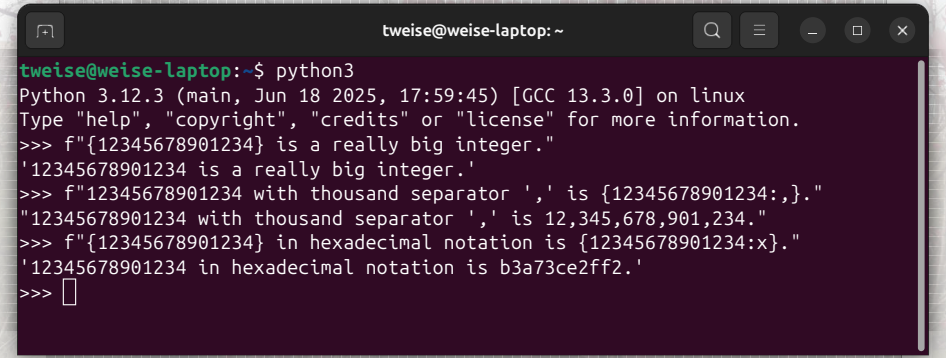
# f-strings

- Wir haben in den Slides zum Datentyp `int` verschiedene Zahlensysteme erwähnt. Fügen wir `:x` an den Ausdruck an, so wird sein Ergebnis als Hexadezimalzahl dargestellt.



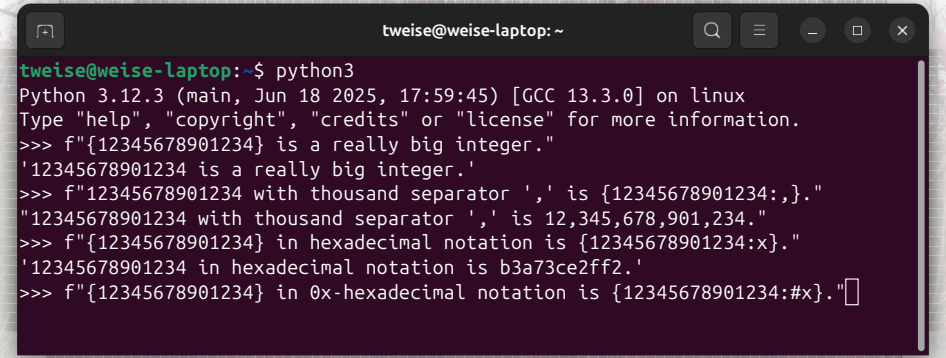
# f-strings

- Wir haben in den Slides zum Datentyp `int` verschiedene Zahlensysteme erwähnt. Fügen wir `:x` an den Ausdruck an, so wird sein Ergebnis als Hexadezimalzahl dargestellt. So wird aus der (Dezimal-)Zahl `12345678901234` die Hexadezimalzahl `b3a73ce2ff2`.



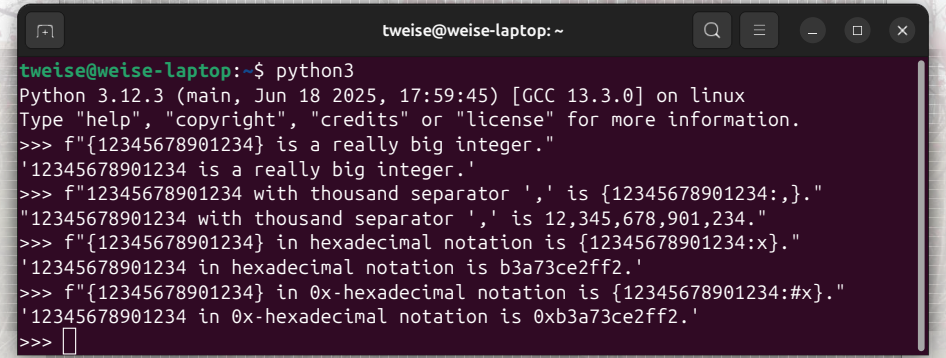
# f-strings

- Wollen wir zusätzlich noch das `0x`-Präfix mit davor haben, schreiben wir `:#x` anstatt von `:x`.



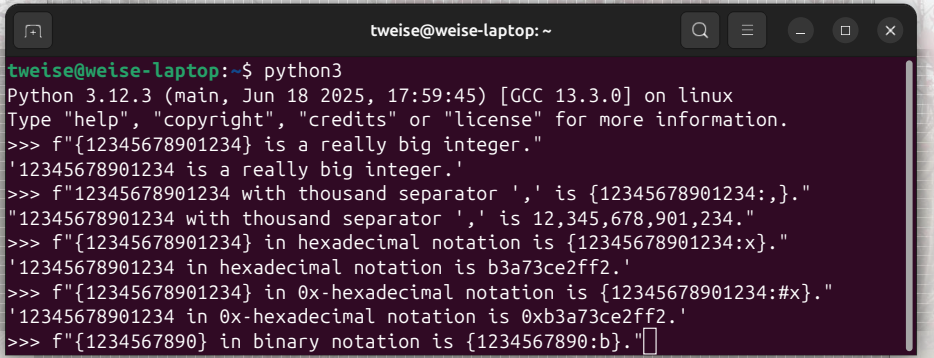
# f-strings

- Wollen wir zusätzlich noch das `0x`-Präfix mit davor haben, schreiben wir `:#x` anstatt von `:x`. So wird aus der (Dezimal-)Zahl `12345678901234` die Hexadezimalzahl `0xb3a73ce2ff2`.



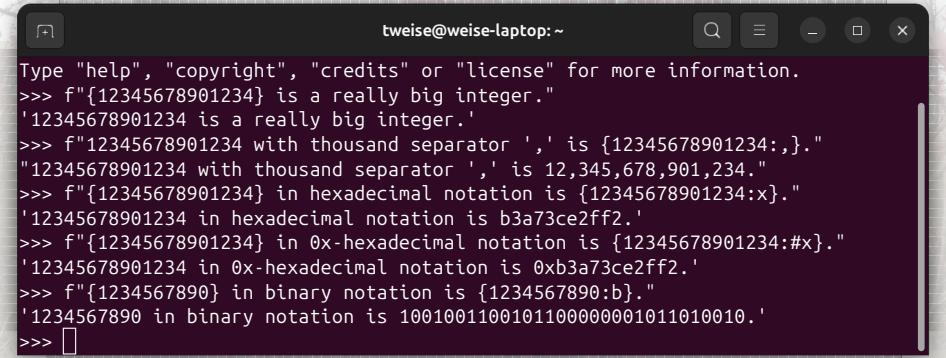
# f-strings

- Fügen wir `:b` an den Ausdruck an, so wird sein Ergebnis als Binärzahl dargestellt.



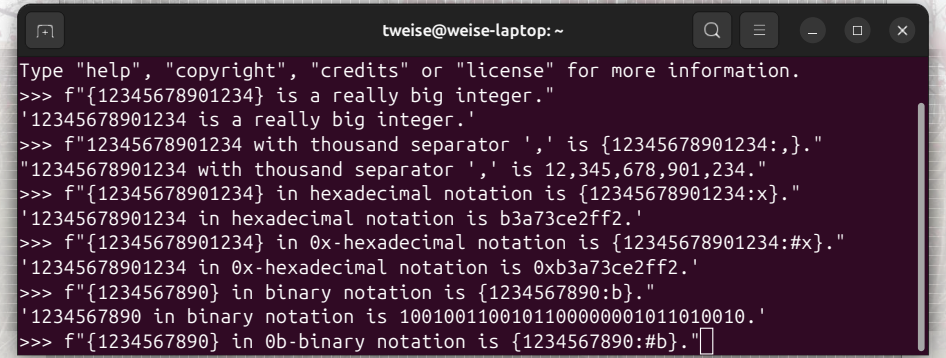
# f-strings

- Fügen wir `:b` an den Ausdruck an, so wird sein Ergebnis als Binärzahl dargestellt. So wird aus der (Dezimal-)Zahl `1234567890` die Binärzahl `1001001100101100000001011010010`.



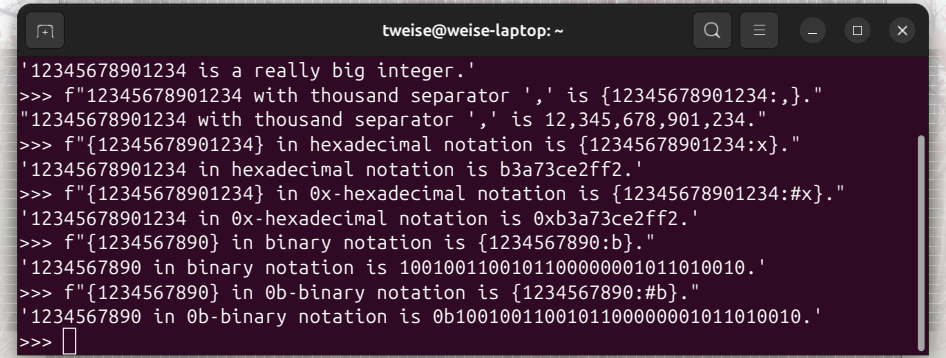
# f-strings

- Fügen wir `:#b` an den Ausdruck an, so wird sein Ergebnis als Binärzahl mit dem Präfix `0b` dargestellt.



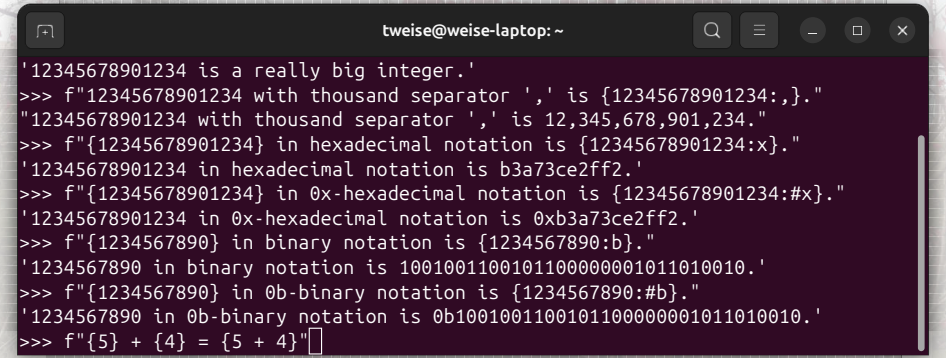
# f-strings

- Fügen wir `:#b` an den Ausdruck an, so wird sein Ergebnis als Binärzahl mit dem Präfix `0b` dargestellt. So wird aus der (Dezimal-)Zahl `1234567890` die Binärzahl `0b1001001100101100000001011010010`.



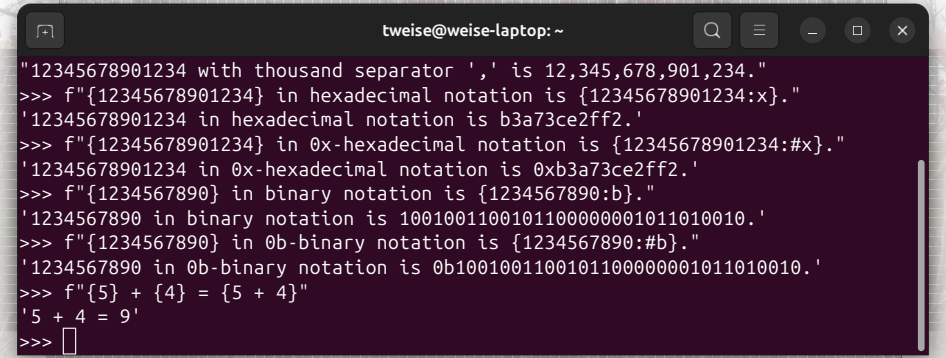
# f-strings

- Nun sind `int`-Zahlen zwar Ausdrücke, aber nicht besonders spannend. Probieren wir es diesmal mit einer echten kleinen Berechnung.



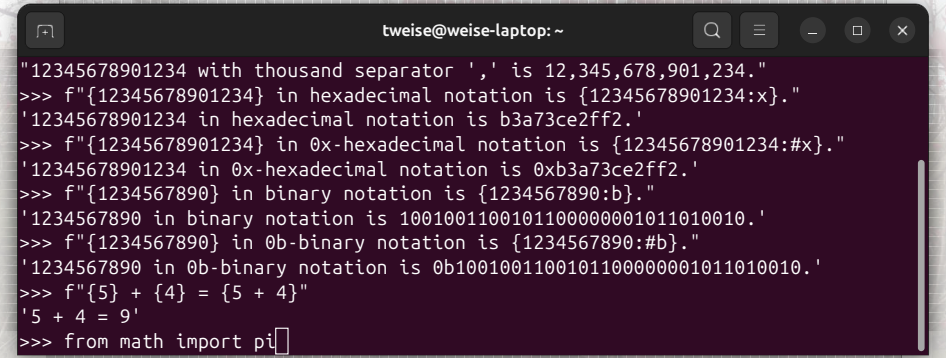
# f-strings

- Nun sind `int`-Zahlen zwar Ausdrücke, aber nicht besonders spannend. Probieren wir es diesmal mit einer echten kleinen Berechnung. `f"{5 + 4}"` wird tatsächlich zu `"9"`.



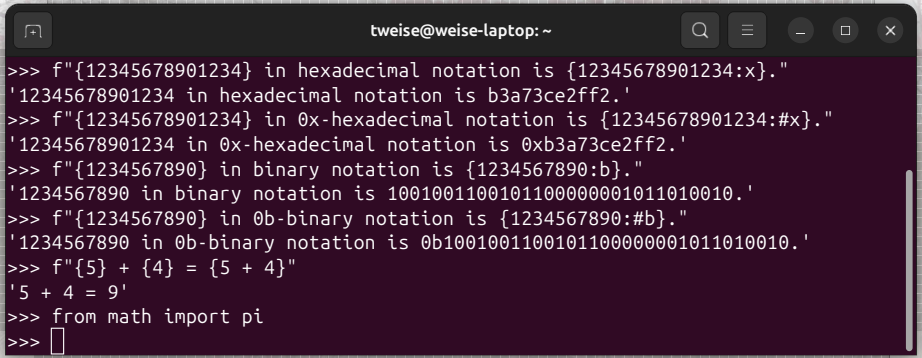
# f-strings

- Wir können in f-Strings auf alle Werte zugreifen, auf die wir aktuell Zugriff haben. Importieren wir also interessehalber mal wieder $\pi$ from `math`-Modul.



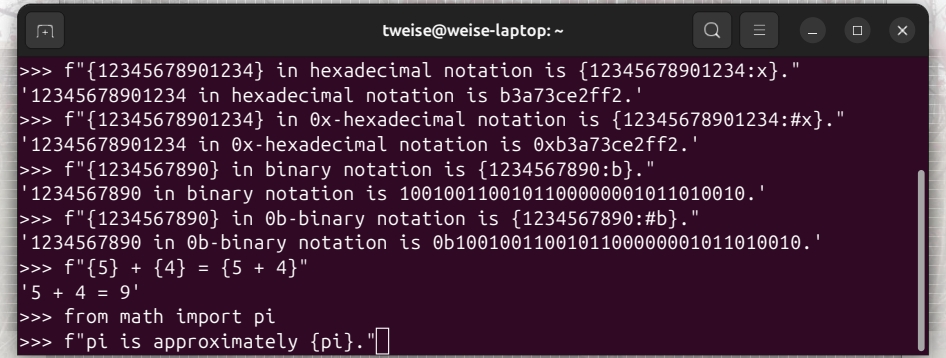
# f-strings

- Wir können in f-Strings auf alle Werte zugreifen, auf die wir aktuell Zugriff haben. Importieren wir also interessehalber mal wieder $\pi$ from `math`-Modul. Was natürlich klappt.



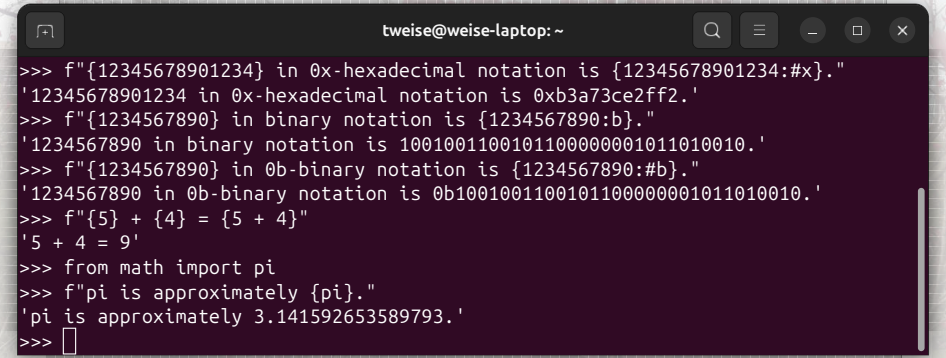
# f-strings

- Nun können wir auch `pi` in f-Strings verwenden.



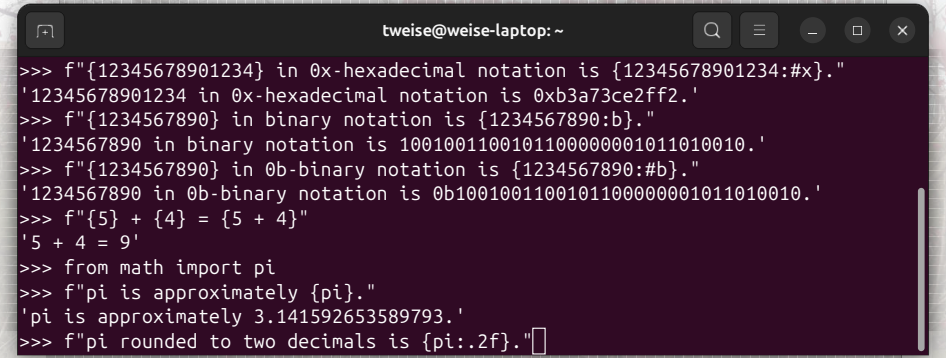
# f-strings

- Nun können wir auch `pi` in f-Strings verwenden. Und sein Wert wird eingefügt.



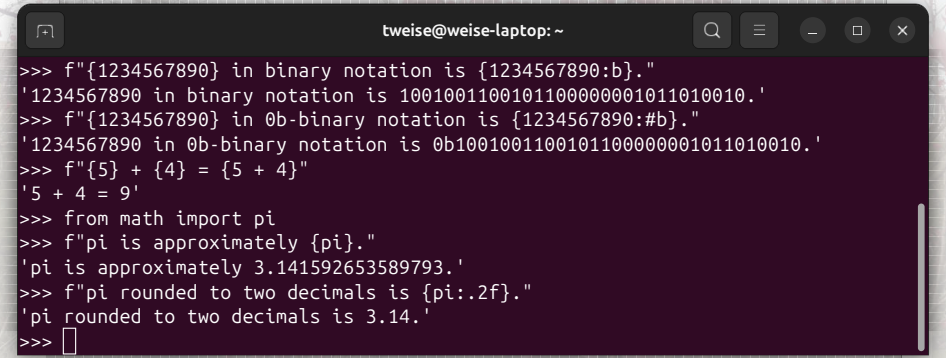
# f-strings

- Wenn wir alle Stellen der `float`-Konstante $\pi$ drucken, dann ist das ziemlich viel. Wollen wir nur `NNN` Nachkommastellen einer Zahl angezeigt bekommen, so fügen wir einfach `:.NNNf` an den Ausdruck an.



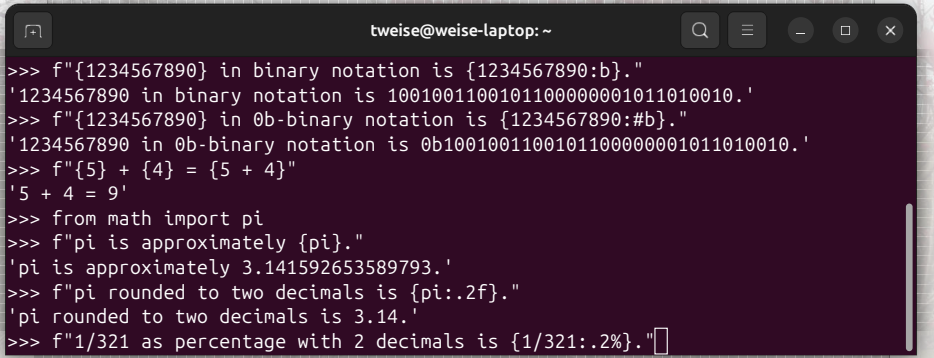
# f-strings

- Wenn wir alle Stellen der `float`-Konstante $\pi$ drucken, dann ist das ziemlich viel. Wollen wir nur `NNN` Nachkommastellen einer Zahl angezeigt bekommen, so fügen wir einfach `:.NNNf` an den Ausdruck an. Das `:.2f` bewirkt, dass der `float`-Wert `pi` auf zwei Nachkommastellen gerundet wird.



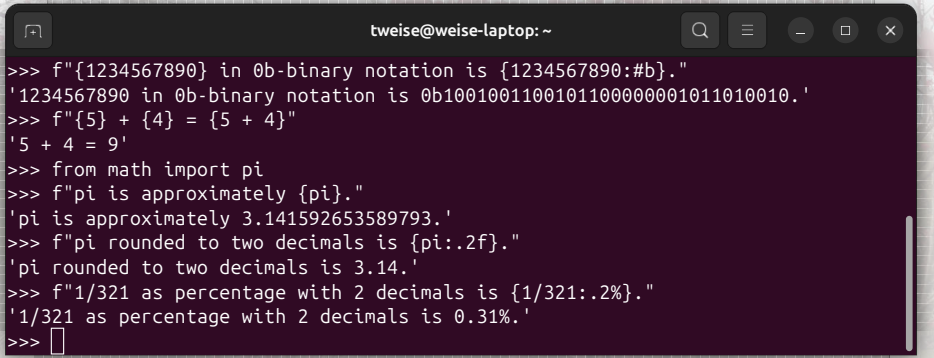
# f-strings

- Wir können eine Zahl in Prozent konvertieren, in dem wir `:%` an den Ausdruck anhängen. Wollen wir den Prozentwert mit `NNN` Nachkommastellen, so hängen wir stattdessen `:.NNN%` an.



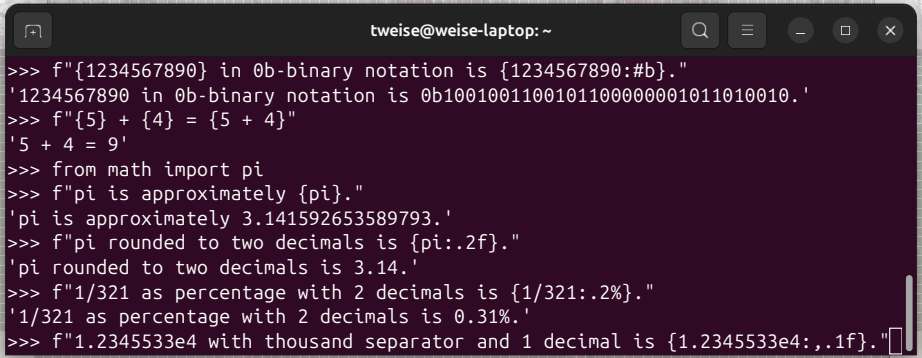
# f-strings

- Wir können eine Zahl in Prozent konvertieren, in dem wir `:%` an den Ausdruck anhängen. Wollen wir den Prozentwert mit `NNN` Nachkommastellen, so hängen wir stattdessen `:.NNN%` an. $\frac{1}{321} \approx 0.00311526$, was $0.311526\%$ entspricht. Die Ausgabe stimmt also.



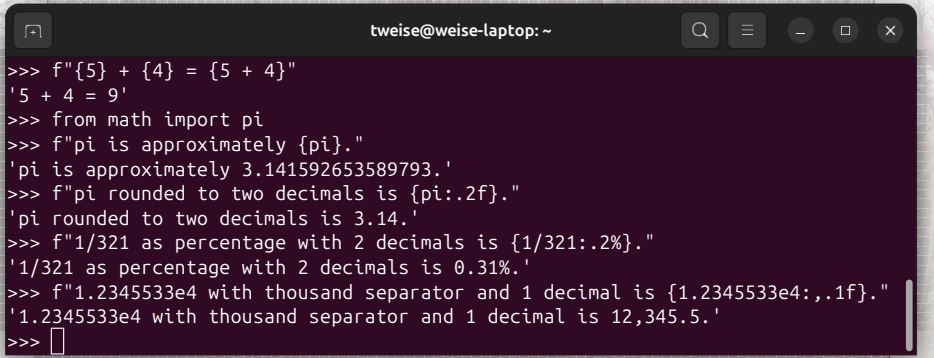
# f-strings

- Wir können auch Tausender-Trennzeichen und Nachkommastellenangaben kombinieren.



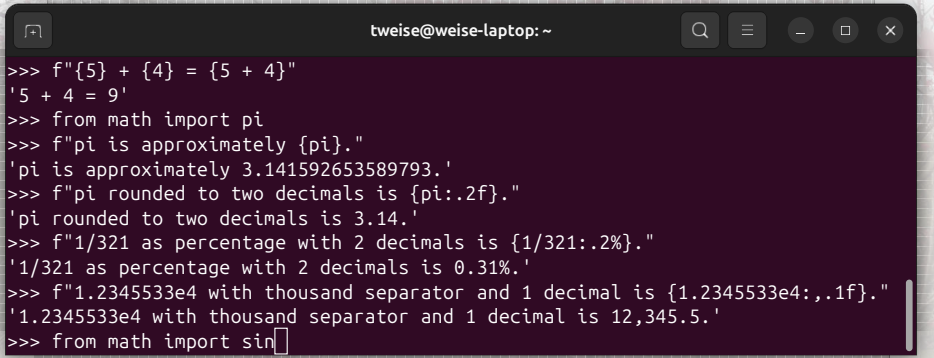
# f-strings

- Wir können auch Tausender-Trennzeichen und Nachkommastellenangaben kombinieren. `1.2345533e4` entspricht $1.2345533 * 10^4$, was wiederum 12345.533 ist. Auf eine Nachkommastelle genau ergibt das 12345.5.



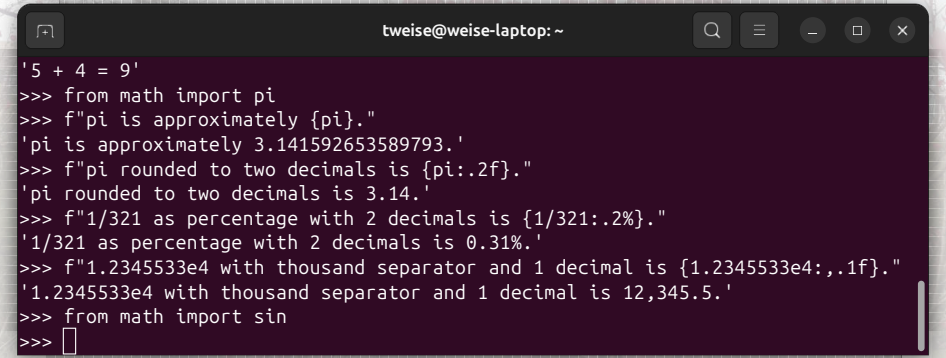
# f-strings

- Importieren wir die Sinus-Funktion aus dem Modul `math`.



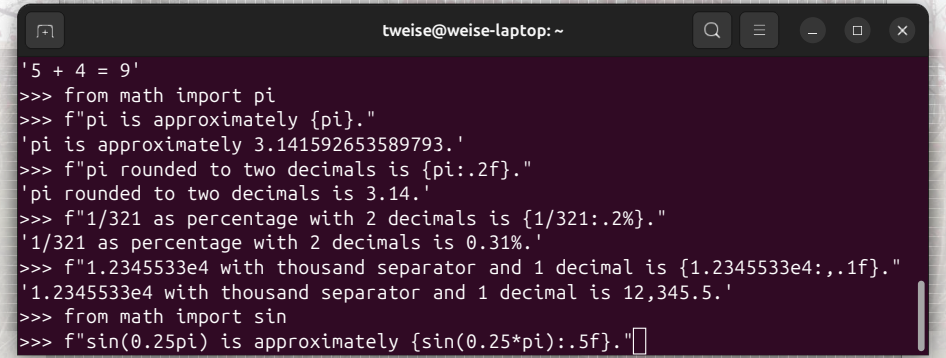
# f-strings

- Importieren wir die Sinus-Funktion aus dem Modul `math`. Gar kein Problem.



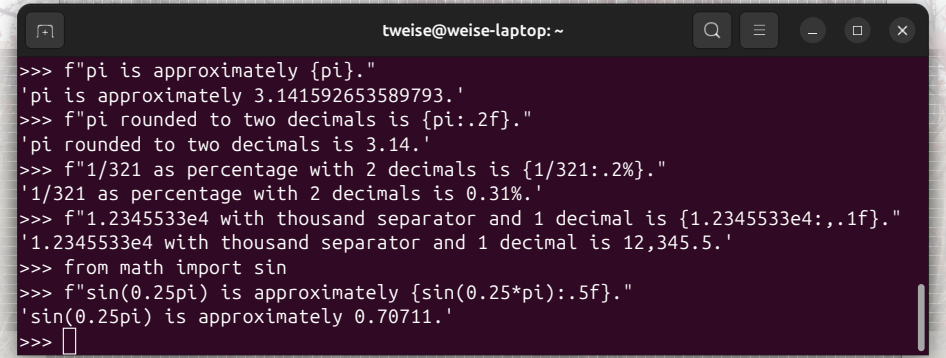
# f-strings

- Und schon können wir sie in f-Strings verwenden, deren Ausgabe wir wieder beliebig formatieren können.



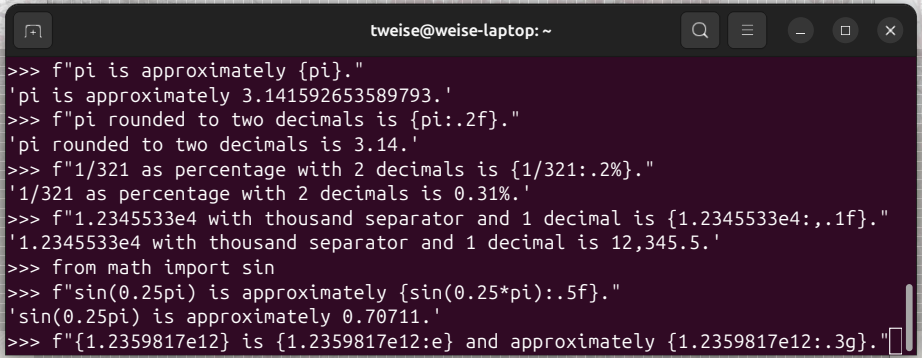
# f-strings

- Und schon können wir sie in f-Strings verwenden, deren Ausgabe wir wieder beliebig formatieren können. $\sin \frac{\pi}{4} = \frac{\sqrt{2}}{2} \approx 0.7071067811865476$.



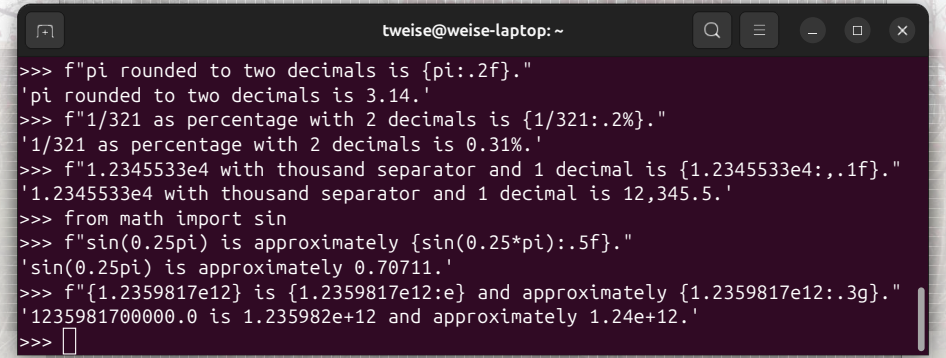
# f-strings

- Das Format `:e` druckt eine Zahl in der wissenschaftlichen Notation. `:.DDDg` benutzt ebenfalls die wissenschaftliche Notation, druckt jedoch nur `DDD` Ziffern (insgesamt, nicht Nachkommastellen).



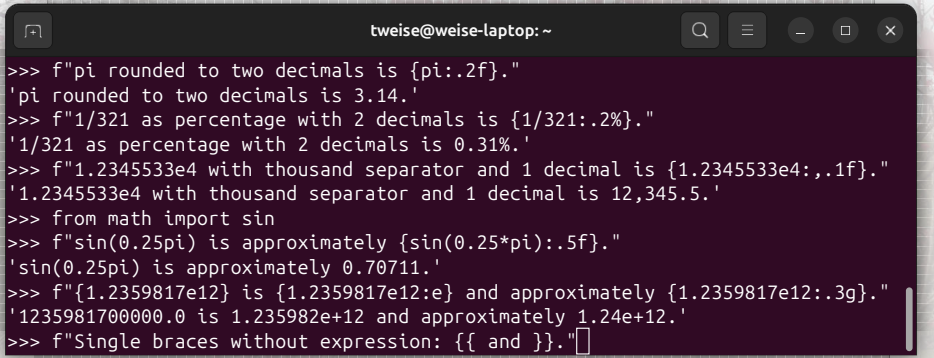
# f-strings

- Das Format `:e` druckt eine Zahl in der wissenschaftlichen Notation. `:.DDDg` benutzt ebenfalls die wissenschaftliche Notation, druckt jedoch nur `DDD` Ziffern (insgesamt, nicht Nachkommastellen). Aus `:.3g` folgert, dass nur drei Ziffern (`1.24`) gedruckt werden.



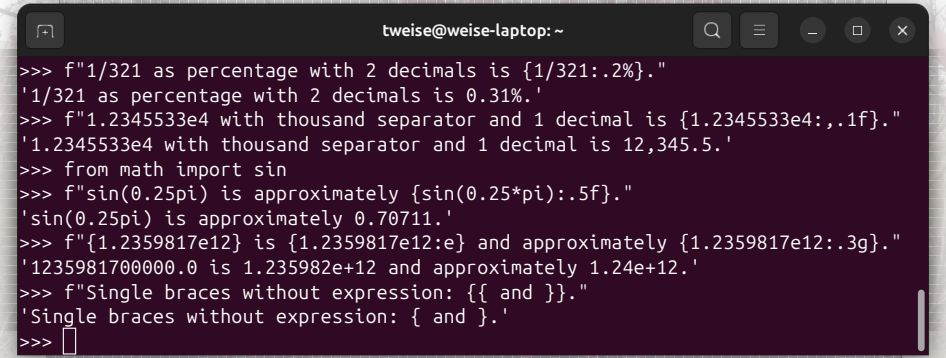
# f-strings

- Wenn die geschwungenen Klammern `{` und `}` Ausdrücke in den Zeichenketten begrenzen ... was machen wir dann, wenn wir mal eine geschwungene Klammer in einem String haben wollen?



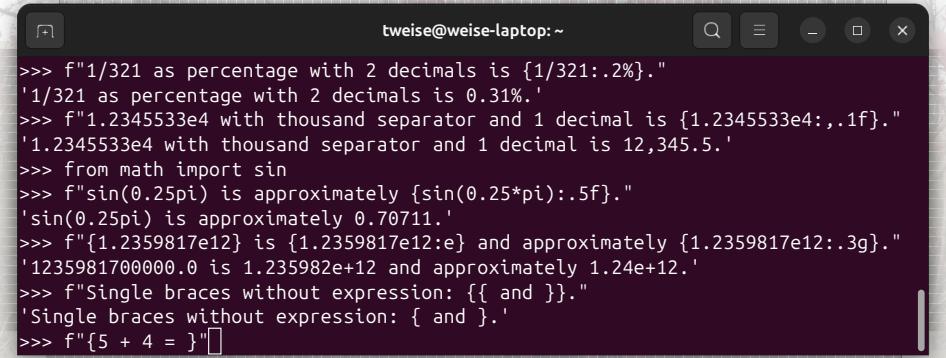
# f-strings

- Wenn die geschwungenen Klammern `{` und `}` Ausdrücke in den Zeichenketten begrenzen ... was machen wir dann, wenn wir mal eine geschwungene Klammer in einem String haben wollen? Einfach *zwei* geschwungene Klammern schreiben.



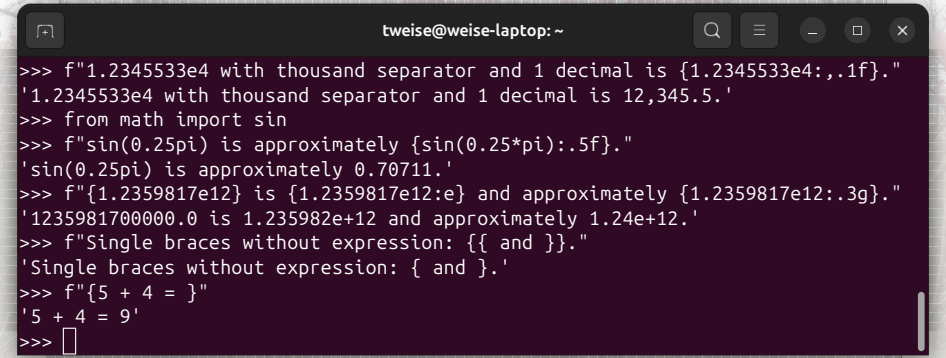
# f-strings

- Ein wirklich cooles Feature ist, dass wir sowohl einen Ausdruck `A` als auch dessen Ergebnis `E` ausdrucken können, in dem wir `f"{A = }"` schreiben.



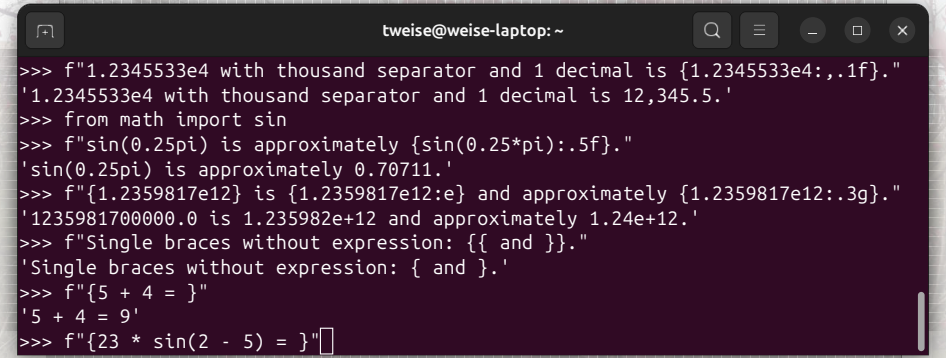
# f-strings

- Ein wirklich cooles Feature ist, dass wir sowohl einen Ausdruck `A` als auch dessen Ergebnis `E` ausdrucken können, in dem wir `f"{A = }"` schreiben. Das wird dann in `"A = E"` umgewandelt.



# f-strings

- Das funktioniert auch mit komplizierten Ausdrücken.



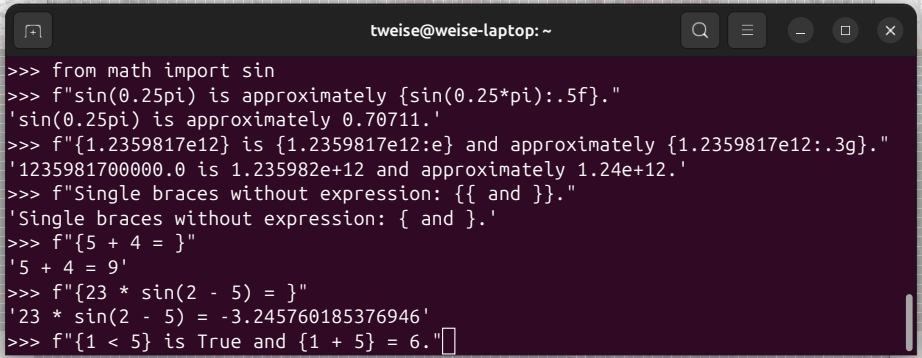
# f-strings

- Das funktioniert auch mit komplizierten Ausdrücken. Problemlos.



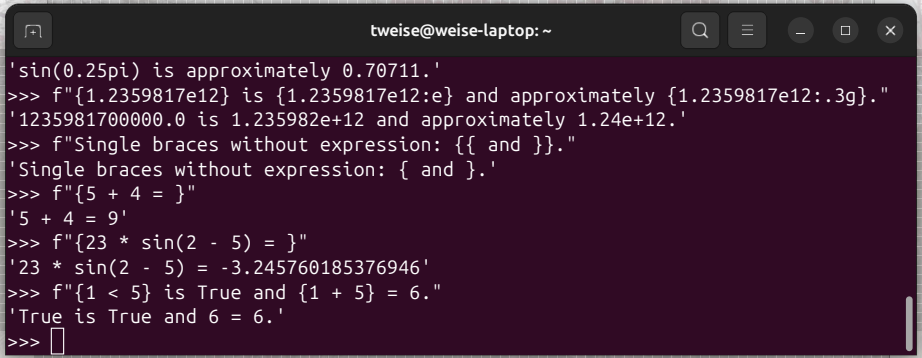
# f-strings

- Kommen wir nun zu unserem Anfangsbeispiel, dem komplizierten String `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."`, zurück. Als f-String sieht das so aus: `f"{1 < 5}is True and {1 + 5}= 6."` Beide Varianten ergeben `"True is True and 6 = 6."`.



# f-strings

- Kommen wir nun zu unserem Anfangsbeispiel, dem komplizierten String `str(1 < 5)+ " is True and "+ str(1 + 5)+ "= 6."`, zurück. Als f-String sieht das so aus: `f"{1 < 5}is True and {1 + 5}= 6."` Beide Varianten ergeben `"True is True and 6 = 6."`.



# f-strings



- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können: f-Strings.
- f-Strings erlauben es uns, die Ergebnisse von Ausdrücken in Zeichenketten einzufügen[6,25,26,28,42,53].
- Sie haben eine einfache Syntax:
  1. Sie sind mit `f"..."` begrenzt (beachten Sie das kleine `f` vor dem einleitenden Anführungszeichen!).
  2. Sie können beliebige Ausdrücke enthalten, die jeweils mit `{...}` begrenzt sind.
  3. Zusätzlich können Markierungen für die Formatierung der Ausdruck-Ergebnisse angegeben werden.
  4. Während der sogenannten String-Interpolation, welche automatisch passiert, werden diese Ausdrücke dann ausgerechnet und ihre Ergebnisse an ihrer Stelle in den Text eingefügt.
- Das waren natürlich nur einige Beispiele.

# f-strings



- In der Tat gibt es eine Möglichkeit, schön formatierte Zeichenketten zu erstellen, die Ergebnisse von Berechnungen beinhalten können: f-Strings.
- f-Strings erlauben es uns, die Ergebnisse von Ausdrücken in Zeichenketten einzufügen[6,25,26,28,42,53].
- Sie haben eine einfache Syntax:
  1. Sie sind mit `f"..."` begrenzt (beachten Sie das kleine `f` vor dem einleitenden Anführungszeichen!).
  2. Sie können beliebige Ausdrücke enthalten, die jeweils mit `{...}` begrenzt sind.
  3. Zusätzlich können Markierungen für die Formatierung der Ausdruck-Ergebnisse angegeben werden.
  4. Während der sogenannten String-Interpolation, welche automatisch passiert, werden diese Ausdrücke dann ausgerechnet und ihre Ergebnisse an ihrer Stelle in den Text eingefügt.
- Das waren natürlich nur einige Beispiele.
- Mehr Informationen gibt es unter *Python f-Strings*[6], "Formatted String Literals"[25], und *Literal String Interpolation*[53].

# Strings in Andere Datentypen Konvertieren



## Strings in Andere Datentypen Konvertieren



- Oft wollen wir Daten in unser Program einlesen oder eingeben.

## Strings in Andere Datentypen Konvertieren



- Oft wollen wir Daten in unser Program einlesen oder eingeben.
- Die Daten sind of als Text gespeichert oder werden als Text in ein Terminal eingegeben.

## Strings in Andere Datentypen Konvertieren



- Oft wollen wir Daten in unser Program einlesen oder eingeben.
- Die Daten sind of als Text gespeichert oder werden als Text in ein Terminal eingegeben.
- Wir brauchen sie dann aber oft als andere Datentypen, z. B. als Zahlen.

## Strings in Andere Datentypen Konvertieren



- Oft wollen wir Daten in unser Program einlesen oder eingeben.
- Die Daten sind of als Text gespeichert oder werden als Text in ein Terminal eingegeben.
- Wir brauchen sie dann aber oft als andere Datentypen, z. B. als Zahlen.
- Viele Datentypen in Python bieten eine Funktion an, die eine Zeichenkette in sie umwandelt.

## Strings in Andere Datentypen Konvertieren



- Oft wollen wir Daten in unser Program einlesen oder eingeben.
- Die Daten sind of als Text gespeichert oder werden als Text in ein Terminal eingegeben.
- Wir brauchen sie dann aber oft als andere Datentypen, z. B. als Zahlen.
- Viele Datentypen in Python bieten eine Funktion an, die eine Zeichenkette in sie umwandelt.
- `int("4")` erzeugt z. B. den `int`-Wert `4`.

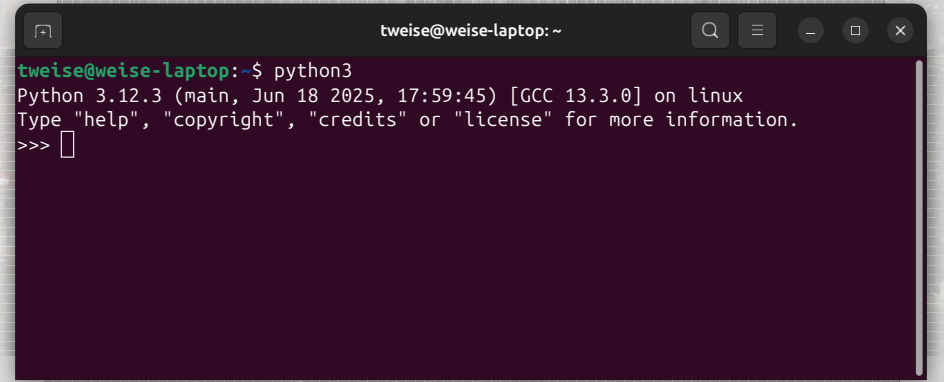
# Strings in Andere Datentypen Konvertieren



- Oft wollen wir Daten in unser Program einlesen oder eingeben.
- Die Daten sind of als Text gespeichert oder werden als Text in ein Terminal eingegeben.
- Wir brauchen sie dann aber oft als andere Datentypen, z. B. als Zahlen.
- Viele Datentypen in Python bieten eine Funktion an, die eine Zeichenkette in sie umwandelt.
- `int("4")` erzeugt z. B. den `int`-Wert `4`.
- Schauen wir uns das mal an.

# Strings in Andere Datentypen Konvertieren

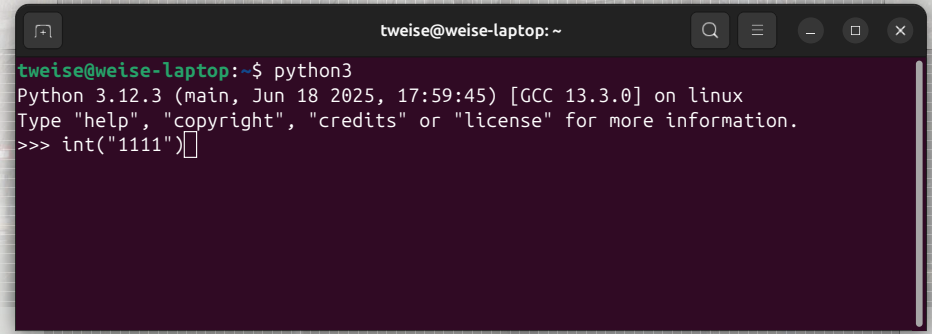
- Schauen wir uns das mal an.



# Strings in Andere Datentypen Konvertieren



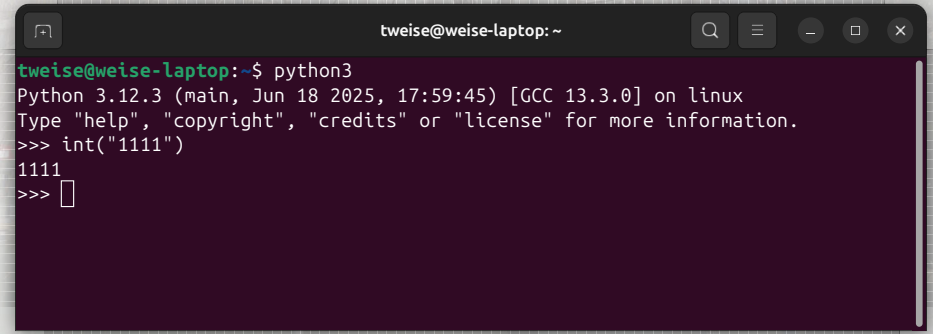
- Die Funktion `int(a)` wandelt einen Wert `a` eines anderen Datentypen in einen `int`-Wert um. `a` könnte z. B. ein String sein.



# Strings in Andere Datentypen Konvertieren

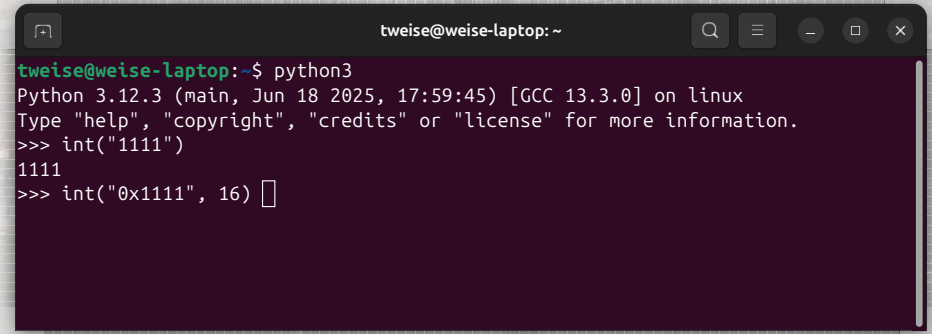


- Die Funktion `int(a)` wandelt einen Wert `a` eines anderen Datentypen in einen `int`-Wert um. `a` könnte z. B. ein String sein. `int("1111")` ergibt daher den Integer `1111`.



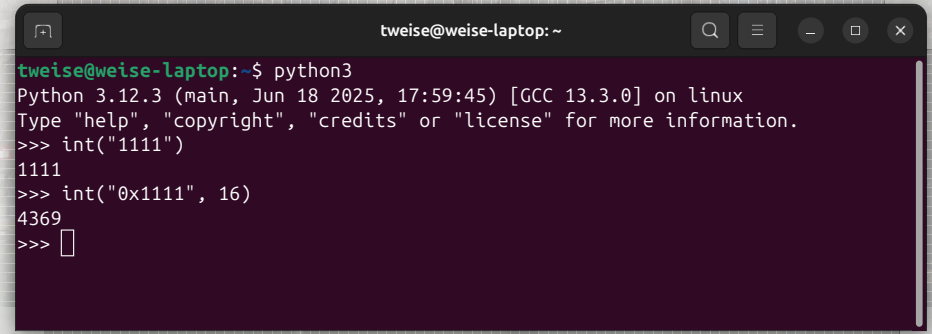
# Strings in Andere Datentypen Konvertieren

- Die Funktion hat einen optionalen zweiten Parameter, der die Basis des zu benutzenden Zahlensystems angibt. Die bekannten Präfixe werden automatisch erkannt.



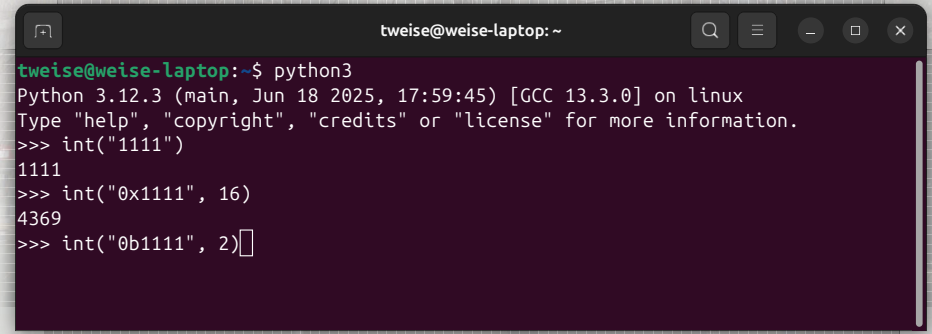
# Strings in Andere Datentypen Konvertieren

- Die Funktion hat einen optionalen zweiten Parameter, der die Basis des zu benutzenden Zahlensystems angibt. Die bekannten Präfixe werden automatisch erkannt.
  `int("0x1111", 16)` wandelt eine Hexdezimalzahl `0x1111` um, es gilt
  $1 * 1 + 1 * 16 + 1 * 16^2 + 1 * 16^3 = 1 + 16 + 256 + 4096 = 4396$ und das ergibt daher den Integer `4369`.



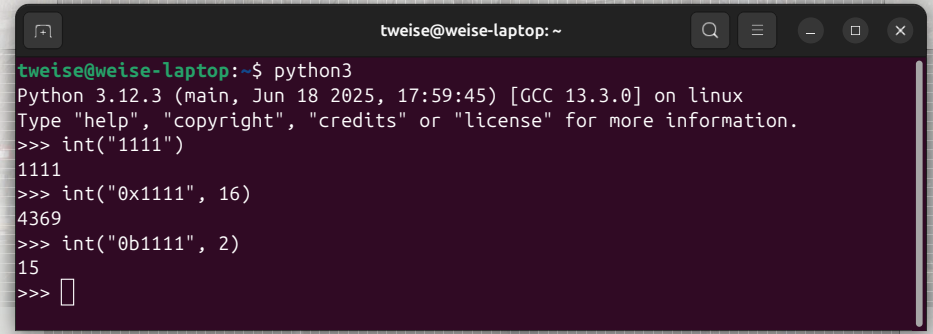
# Strings in Andere Datentypen Konvertieren

- Die Funktion hat einen optionalen zweiten Parameter, der die Basis des zu benutzenden Zahlensystems angibt. Die bekannten Präfixe werden automatisch erkannt.



# Strings in Andere Datentypen Konvertieren

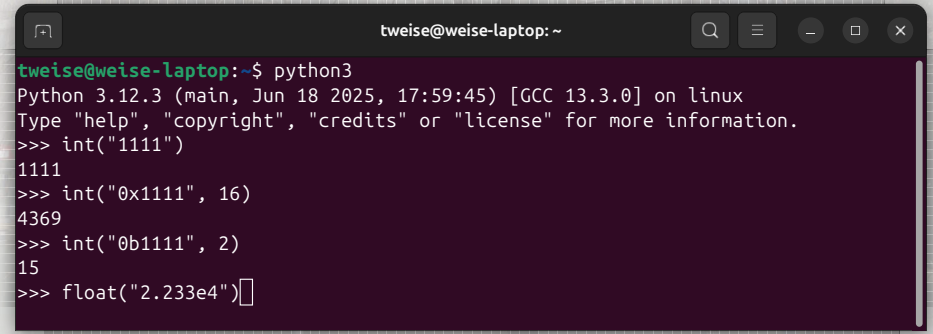
- Die Funktion hat einen optionalen zweiten Parameter, der die Basis des zu benutzenden Zahlensystems angibt. Die bekannten Präfixe werden automatisch erkannt.
  `int("0b1111", 2)` wandelt eine Binärzahl `0b1111` um, es gilt
  $1 * 1 + 1 * 2 + 1 * 2^2 + 1 * 2^3 = 1 + 2 + 4 + 8 = 15$ und das ergibt daher den Integer `15`.



# Strings in Andere Datentypen Konvertieren

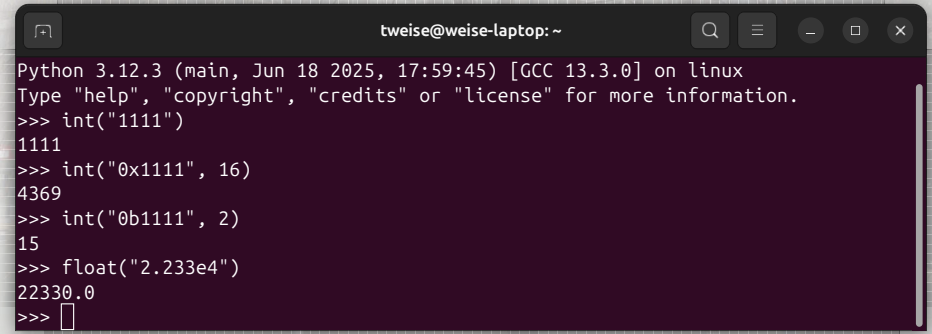


- Die Funktion `float` wandelt einen Wert `a` eines anderen Datentypen in einen `float`-Wert um. `a` könnte z. B. ein String sein.



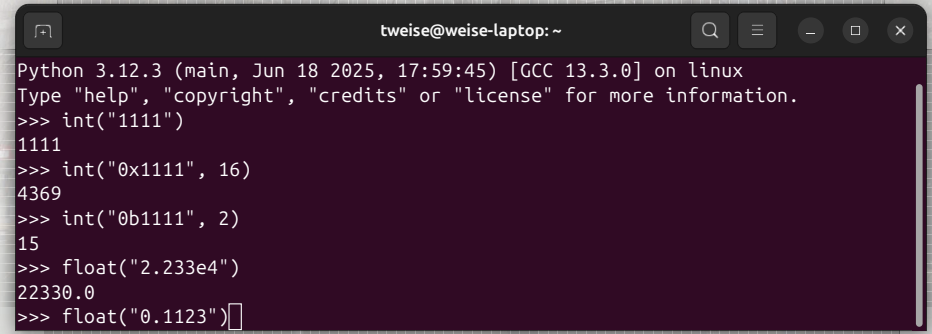
# Strings in Andere Datentypen Konvertieren

- Die Funktion `float` wandelt einen Wert `a` eines anderen Datentypen in einen `float`-Wert um. `a` könnte z. B. ein String sein. `float` erkennt die wissenschaftliche Notation und `float("2.233e4")` wird daher `22330.0`.



# Strings in Andere Datentypen Konvertieren

- Natürlich geht das auch ohne wissenschaftliche Notation. `float("0.1123")` wird zu `0.1123`.



# Strings in Andere Datentypen Konvertieren

- Natürlich geht das auch ohne wissenschaftliche Notation. `float("0.1123")` wird zu `0.1123`.



# Strings in Andere Datentypen Konvertieren



- Es werden auch die Konstanten für „zu groß für `float`“ und „keine Zahl“ erkannt. `float("inf")` wird zu `math.inf`.

```
tweise@weise-laptop: ~
>>> int("1111")
1111
>>> int("0x1111", 16)
4369
>>> int("0b1111", 2)
15
>>> float("2.233e4")
22330.0
>>> float("0.1123")
0.1123
>>> float("inf")
```

# Strings in Andere Datentypen Konvertieren

- Es werden auch die Konstanten für „zu groß für `float`“ und „keine Zahl“ erkannt. `float("inf")` wird zu `math.inf`.



# Strings in Andere Datentypen Konvertieren

- Es werden auch die Konstanten für „zu groß für `float`“ und „keine Zahl“ erkannt.



# Strings in Andere Datentypen Konvertieren

- Es werden auch die Konstanten für „zu groß für `float`“ und „keine Zahl“ erkannt. Und `float("nan")` wird zu `math.nan`.



# Strings in Andere Datentypen Konvertieren



- Die Funktion `bool` wandelt einen Wert `a` eines anderen Datentypen in einen `bool`-Wert um. `a` könnte z. B. ein String sein.

tweise@weise-laptop: ~

```
>>> int("0b1111", 2)
15
>>> float("2.233e4")
22330.0
>>> float("0.1123")
0.1123
>>> float("inf")
inf
>>> float("nan")
nan
>>> bool("True")
```

# Strings in Andere Datentypen Konvertieren



- Die Funktion `bool` wandelt einen Wert `a` eines anderen Datentypen in einen `bool`-Wert um. `a` könnte z. B. ein String sein. Aus `bool("True")` wird daher `True`.

```
tweise@weise-laptop: ~
>>> float("2.233e4")
22330.0
>>> float("0.1123")
0.1123
>>> float("inf")
inf
>>> float("nan")
nan
>>> bool("True")
True
>>>
```

# Strings in Andere Datentypen Konvertieren



- Die Funktion `bool` wandelt einen Wert `a` eines anderen Datentypen in einen `bool`-Wert um. `a` könnte z. B. ein String sein.



# Strings in Andere Datentypen Konvertieren

- Die Funktion `bool` wandelt einen Wert `a` eines anderen Datentypen in einen `bool`-Wert um. `a` könnte z. B. ein String sein. Und `bool("False")` wird zu `False`.



# Strings in Andere Datentypen Konvertieren

- Die Funktion `bool` wandelt einen Wert `a` eines anderen Datentypen in einen `bool`-Wert um. `a` könnte z. B. ein String sein. Und `bool("False")` wird zu `False`. Halt!



# Strings in Andere Datentypen Konvertieren

- Die Funktion `bool` wandelt einen Wert `a` eines anderen Datentypen in einen `bool`-Wert um. `a` könnte z. B. ein String sein. Und `bool("False")` wird zu `False`. Halt! Nein wird es nicht!



# Strings in Andere Datentypen Konvertieren

- Die Funktion `bool` wandelt einen Wert `a` eines anderen Datentypen in einen `bool`-Wert um. `a` könnte z. B. ein String sein. Und `bool("False")` wird auch zu `True`!



# Strings in Andere Datentypen Konvertieren

- Die Funktion `bool` wandelt einen Wert `a` eines anderen Datentypen in einen `bool`-Wert um. `a` könnte z. B. ein String sein. Und `bool("False")` wird auch zu `True`!
- Der Grund ist, dass die Funktion `bool` in so genanntes truth value testing durchführt[7,66].



# Strings in Andere Datentypen Konvertieren



- Der Grund ist, dass die Funktion `bool` in so genanntes truth value testing durchführt[7,66].
- Objekte bekommen dabei den Wert `True`, **es sei denn**, sie bieten selber eine explizite Umrechnung in den Typ `bool` an (und die ergibt `False`) oder sie haben eine Länge (und die ist 0).



# Strings in Andere Datentypen Konvertieren

- Objekte bekommen dabei den Wert `True`, **es sei denn**, sie bieten selber eine explizite Umrechnung in den Typ `bool` an (und die ergibt `False`) oder sie haben eine Länge (und die ist 0).
- `"False"` ist ein String mit einer Länge größer als 0.



# Strings in Andere Datentypen Konvertieren

- Objekte bekommen dabei den Wert `True`, **es sei denn**, sie bieten selber eine explizite Umrechnung in den Typ `bool` an (und die ergibt `False`) oder sie haben eine Länge (und die ist 0).
- `"False"` ist ein String mit einer Länge größer als 0. Daher ist `bool("False")` `True`.



# Strings in Andere Datentypen Konvertieren

- Welchen Wahrheitswert hat dann der leere String `""`?



# Strings in Andere Datentypen Konvertieren



- Welchen Wahrheitswert hat dann der leere String `""`? Der wird dann wirklich zu `False`.



# Strings in Andere Datentypen Konvertieren

- Für einen beliebigen nicht-leeren String, z. B. `"blabla"`...



# Strings in Andere Datentypen Konvertieren

- Für einen beliebigen nicht-leeren String, z. B. `"blabla"` gibt `bool` uns aber immer `True`.



# Strings in Andere Datentypen Konvertieren



- Oft wollen wir Daten in unser Program einlesen oder eingeben.
- Die Daten sind of als Text gespeichert oder werden als Text in ein Terminal eingegeben.
- Wir brauchen sie dann aber oft als andere Datentypen, z. B. als Zahlen.
- Viele Datentypen in Python bieten eine Funktion an, die eine Zeichenkette in sie umwandelt.
- Von der Überraschung mit `bool` abgesehen war das relativ einfach.



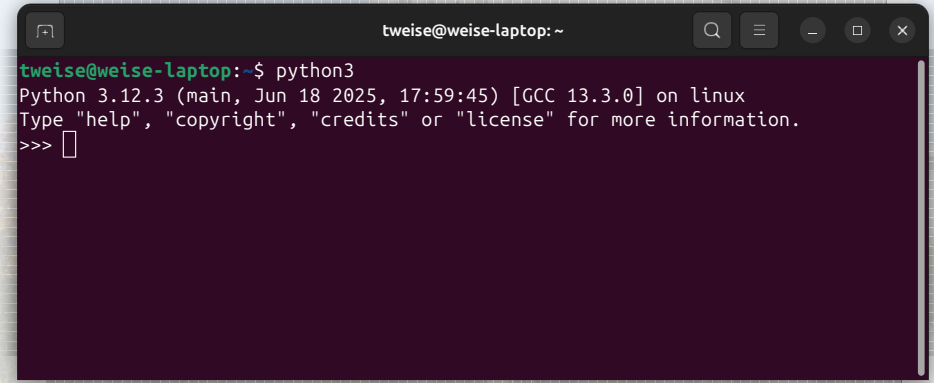
# Strings Escaping

## Strings Escaping

- Zeichenketten werden durch Anführungszeichen begrenzt.

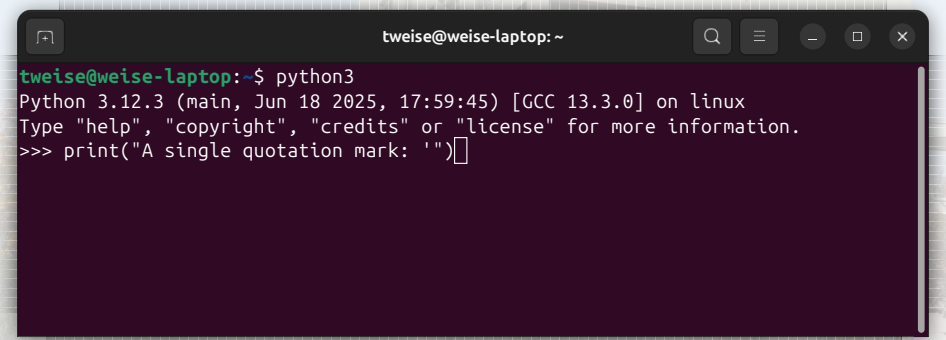
# Strings Escaping

- Zeichenketten werden durch Anführungszeichen begrenzt.
- Was machen wir, wenn wir Anführungszeichen in einer Zeichenkette haben wollen?



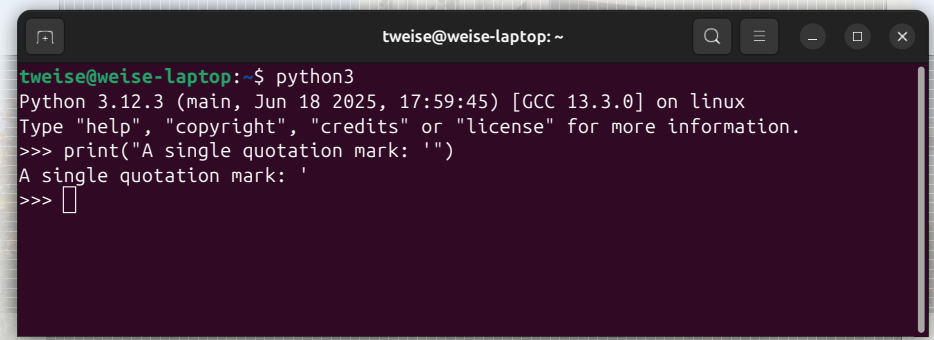
# Strings Escaping

- Wenn wir unsere Strings durch doppelte Anführungszeichen (`"`) begrenzen, dann können problemlos einfache Anführungszeichen (`'`) einfügren.



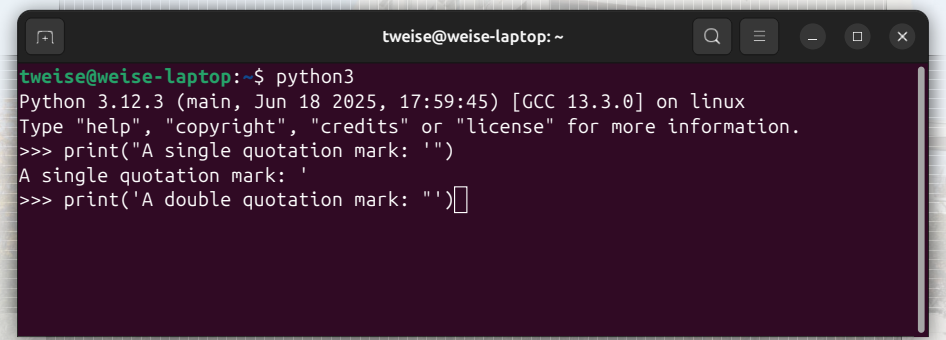
# Strings Escaping

- Wenn wir unsere Strings durch doppelte Anführungszeichen (`"`) begrenzen, dann können problemlos einfache Anführungszeichen (`'`) einfügren. Beachten Sie, dass wenn wir Strings mit `print` ausgeben, die begrenzenden Anführungszeichen nicht in der Ausgabe enthalten sind.



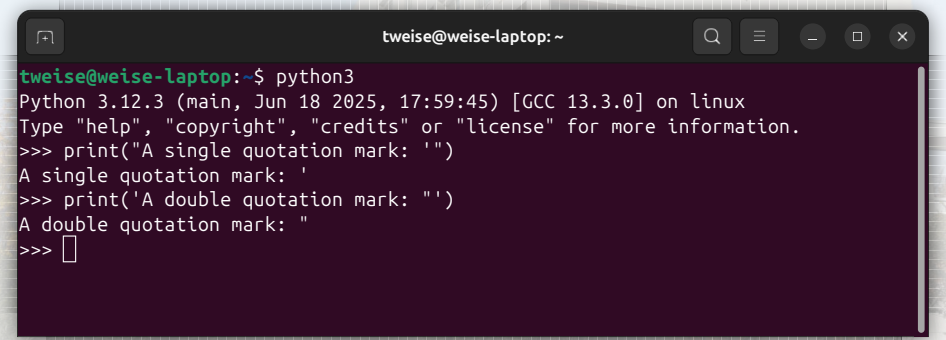
# Strings Escaping

- Wenn wir doppelte Anführungszeichen (`"`) brauchen, dann begrenzen wir unsere Strings eben mit einfachen Anführungszeichen (`'`).



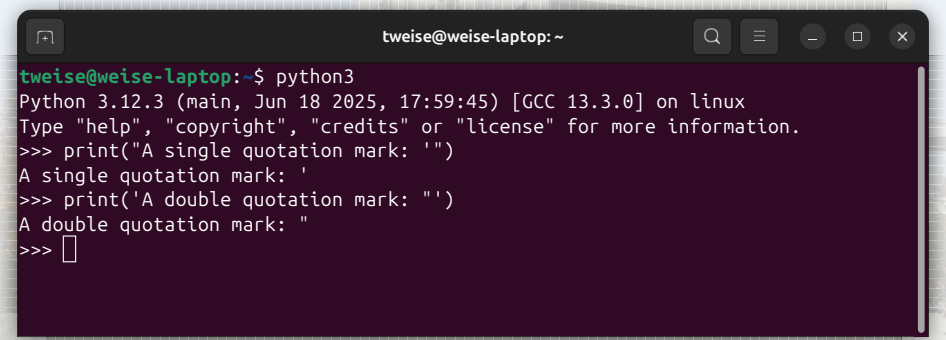
# Strings Escaping

- Wenn wir doppelte Anführungszeichen (`"`) brauchen, dann begrenzen wir unsere Strings eben mit einfachen Anführungszeichen (`'`). Beachten Sie, dass wenn wir Strings mit `print` ausgeben, die begrenzenden Anführungszeichen nicht in der Ausgabe enthalten sind.



# Strings Escaping

- Aber was machen wir, wenn wir beide Arten von Anführungszeichen brauchen?



## Strings Escaping

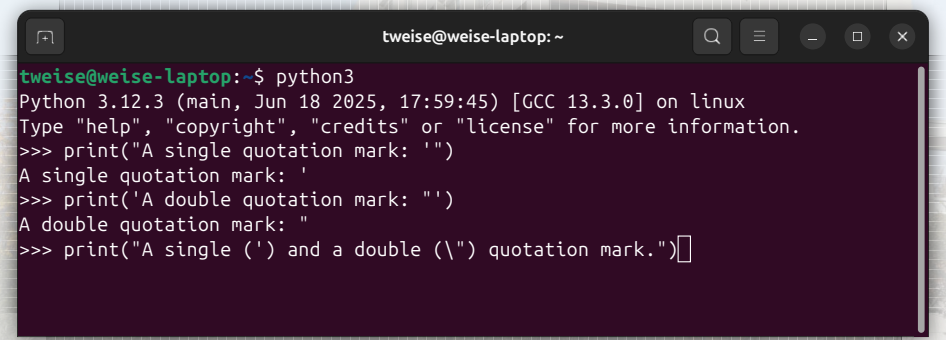
- Aber was machen wir, wenn wir beide Arten von Anführungszeichen brauchen? Dann können wir nicht auf „andere“ Anführungszeichen zur Begrenzung der Strings ausweichen.

## Strings Escaping

- Aber was machen wir, wenn wir beide Arten von Anführungszeichen brauchen? Dann können wir nicht auf „andere“ Anführungszeichen zur Begrenzung der Strings ausweichen.
- Dann brauche wir sogenannte Escape-Sequenzen[19].

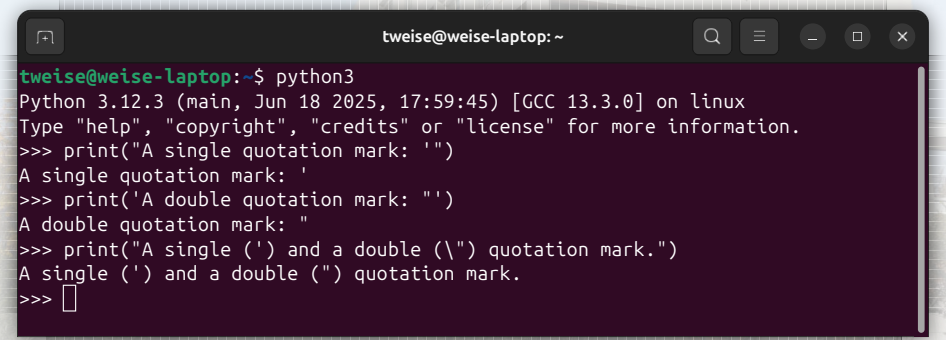
# Strings Escaping

- Wir können dem doppelten Anführungszeichen einfach in Backslash („\“) voranstellen.



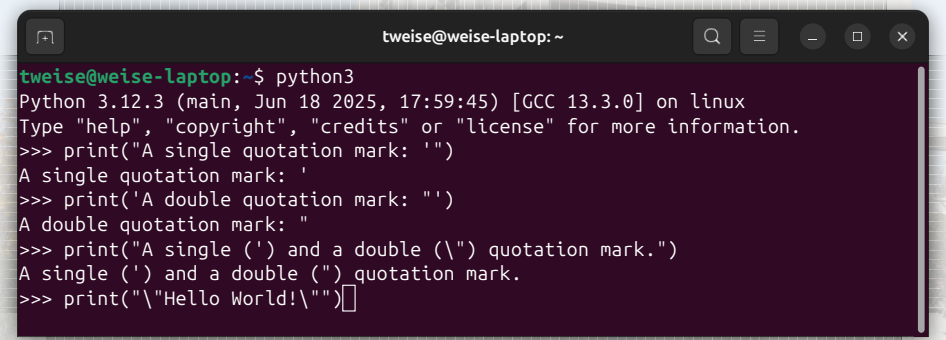
# Strings Escaping

- Wir können dem doppelten Anführungszeichen einfach in Backslash („\“) voranstellen. Dann wird es nicht mehr als String-Begrenzer interpretiert und verwandelt sich in ein normales doppeltes Anführungszeichen.



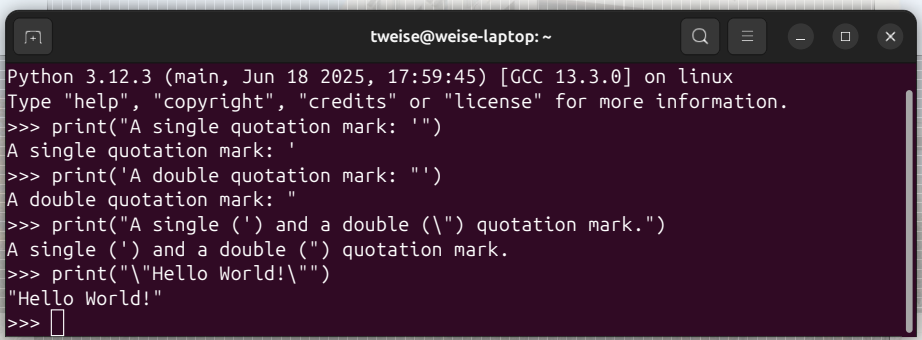
# Strings Escaping

- So können wir bequem doppelte Anführungszeichen in Strings haben, die mit doppelten Anführungszeichen begrenzt sind.



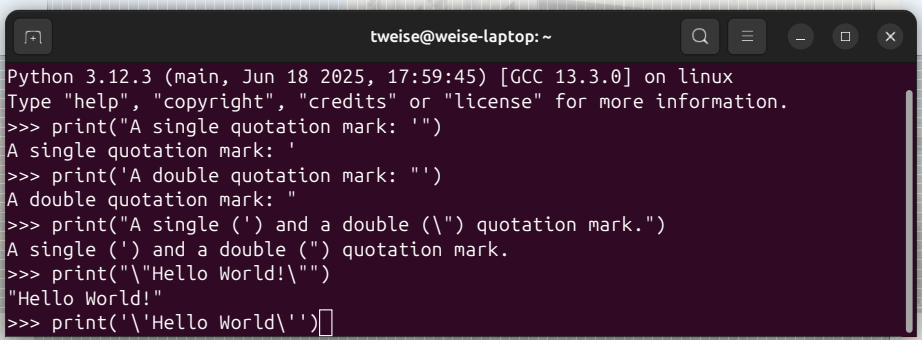
# Strings Escaping

- So können wir bequem doppelte Anführungszeichen in Strings haben, die mit doppelten Anführungszeichen begrenzt sind. Die Backslashes verschwinden dann in der Ausgabe.



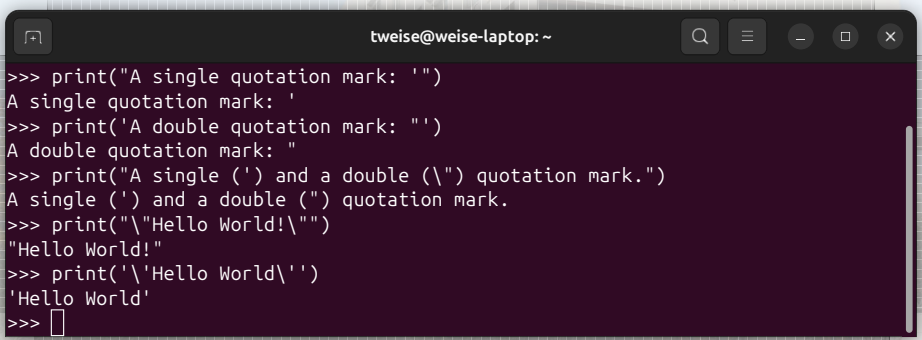
# Strings Escaping

- So können wir bequem einfache Anführungszeichen in Strings haben, die mit einfache Anführungszeichen begrenzt sind. Die Backslashes verschwinden dann in der Ausgabe.



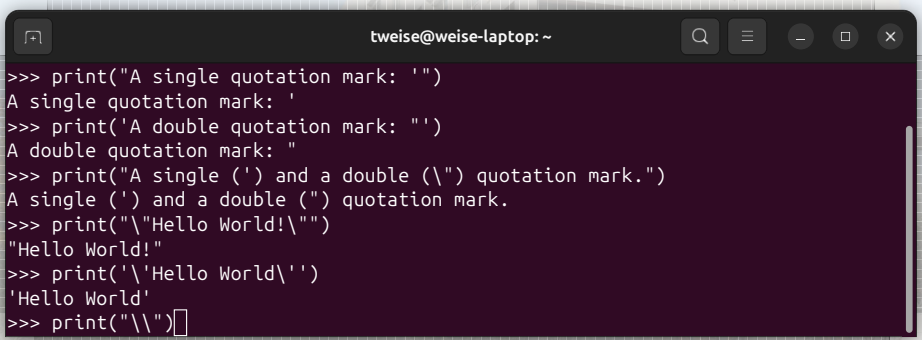
# Strings Escaping

- So können wir bequem einfache Anführungszeichen in Strings haben, die mit einfache Anführungszeichen begrenzt sind. Die Backslashes verschwinden dann in der Ausgabe.



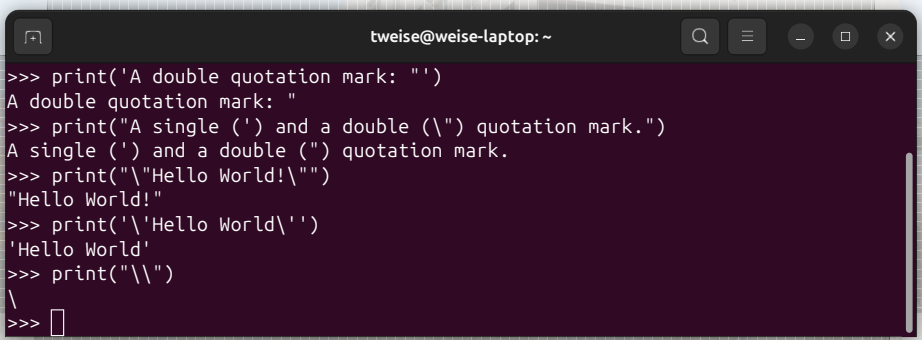
# Strings Escaping

- Aber was machen wir, wenn wir ein Backslash („\“) in einem String haben wollen?



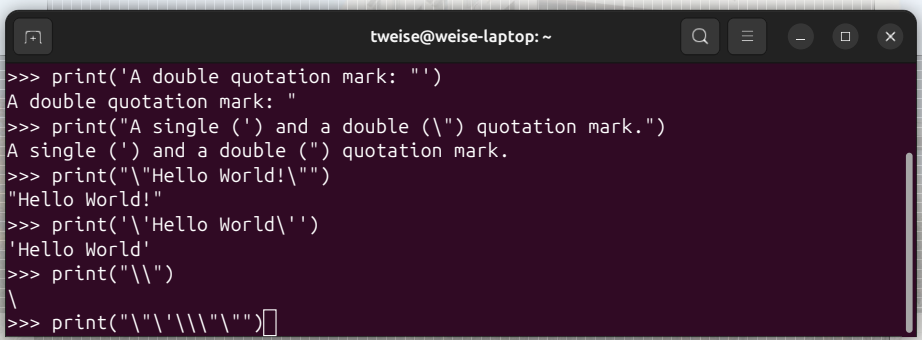
# Strings Escaping

- Aber was machen wir, wenn wir ein Backslash („\“) in einem String haben wollen? Einfach noch ein Backslash davor schreiben. Aus dem doppelten Backslash wird dann ein einfaches Backslash.



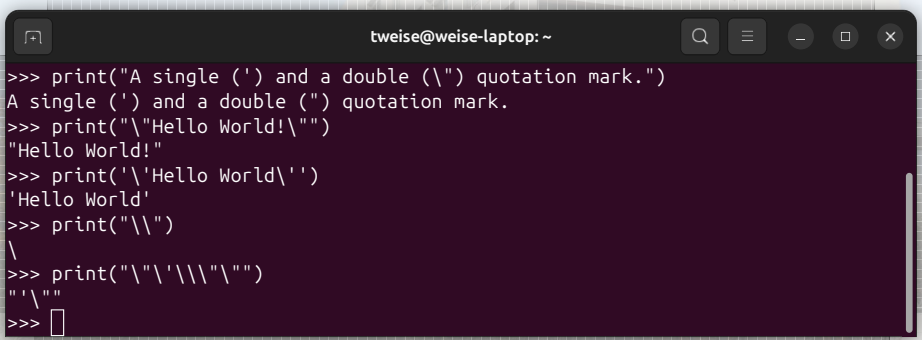
# Strings Escaping

- Natürlich können wir beliebige viele Escape-Sequenzen in unseren Strings verwenden.



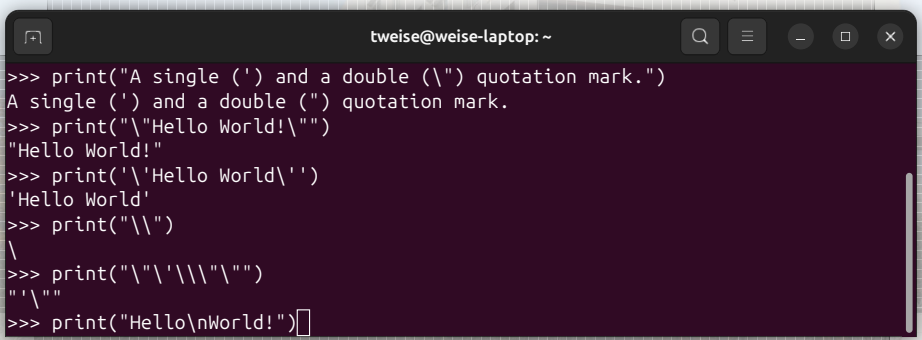
# Strings Escaping

- Natürlich können wir beliebige viele Escape-Sequenzen in unseren Strings verwenden. Der String `"\"\'\\\"\""` beginnt und endet mit `"`. Das erste Zeichen ist ein durch `\"` escapedes `"`. Darauf folget ein escapedes einfaches Anführungszeichen, ein escapedes Backslash, ein zweites escapedes doppeltes Anführungszeichen, und ein drittes escapedes doppeltes Anführungszeichen.



# Strings Escaping

- Die Sequenz `\n` steht für „Newline“, also den Anfang einer neuen Zeile.



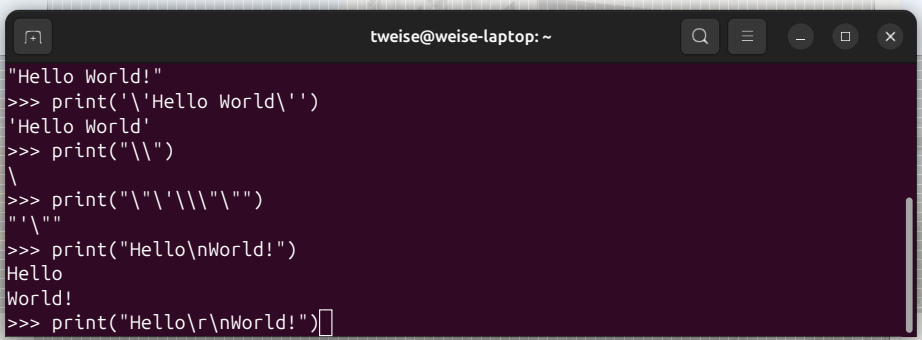
# Strings Escaping

- Die Sequenz `\n` steht für „Newline“, also den Anfang einer neuen Zeile. Somit können wir mehrzeilige Strings in einer einzigen Zeile schreiben.



# Strings Escaping

- Unter Windows kann man auch die Sequenz `\r\n` verwenden.



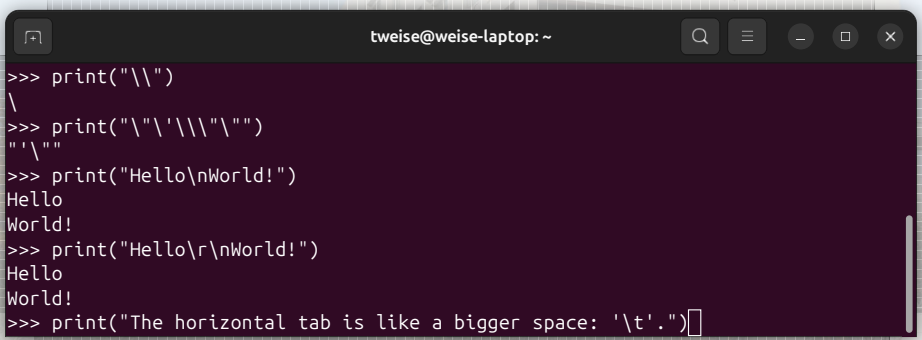
# Strings Escaping

- Unter Windows kann man auch die Sequenz `\r\n` verwenden. Muss man aber nicht, Python gibt auch mit `\n` den richtigen Text aus.



# Strings Escaping

- Die Sequenz `\t` steht für „Tabulator“, welcher oft equivalent zu mehreren (oft 4) Leerzeichen ausgegeben wird. Das braucht man selten.



# Strings Escaping

- Die Sequenz `\t` steht für „Tabulator“, welcher oft equivalent zu mehreren (oft 4) Leerzeichen ausgegeben wird. Das braucht man selten.



# Strings Escaping

- Wollen wir einen String über mehrere Zeilen hinweg schreiben, dann können wir ans Ende der Zeile einfach ein Backslash („\“) schreiben.



# Strings Escaping

- Wollen wir einen String über mehrere Zeilen hinweg schreiben, dann können wir ans Ende der Zeile einfach ein Backslash („\“) schreiben. Wenn wir dann ↵ drücken, geht es in der nächsten Zeile einfach weiter, die im Interpreter dann mit drei Punkten beginnt.



# Strings Escaping

- Wollen wir einen String über mehrere Zeilen hinweg schreiben, dann können wir ans Ende der Zeile einfach ein Backslash („\“) schreiben. Wenn wir dann ↵ drücken, geht es in der nächsten Zeile einfach weiter, die im Interpreter dann mit drei Punkten beginnt. So können wir den Text weiterschreiben.



# Strings Escaping

- Wollen wir einen String über mehrere Zeilen hinweg schreiben, dann können wir ans Ende der Zeile einfach ein Backslash („\“) schreiben. Wenn wir dann ↵ drücken, geht es in der nächsten Zeile einfach weiter, die im Interpreter dann mit drei Punkten beginnt. So können wir den Text weiterschreiben. Der String beinhaltet dann keinen Zeilenumbruch.



# Strings Escaping

- Zeichenketten werden durch Anführungszeichen begrenzt.
- Was machen wir, wenn wir Anführungszeichen in einer Zeichenkette haben wollen?
- Aber was machen wir, wenn wir beide Arten von Anführungszeichen brauchen? Dann können wir nicht auf „andere“ Anführungszeichen zur Begrenzung der Strings ausweichen.
- Dann brauche wir sogenannte Escape-Sequenzen[19].
- Wir können dem doppelten Anführungszeichen einfach in Backslash („\“) voranstellen. Dann wird es nicht mehr als String-Begrenzer interpretiert und verwandelt sich in ein normales doppeltes Anführungszeichen.
- Escape-Sequenzen sind also nützliche Werkzeuge, um Zeichen in Strings zu schreiben, die wir sonst nicht schreiben könnten.

# Mehrzeilige Strings



## Mehrzeilige Strings



- Wir können Strings erstellen, die über mehrere Zeilen gehen, in dem wir die Escape-Sequenz `\n` einfügen.

## Mehrzeilige Strings




- Wir können Strings erstellen, die über mehrere Zeilen gehen, in dem wir die Escape-Sequenz `\n` einfügen.
- Wenn die Strings lang werden, dann kann das aber zu häßlichen sehr langen Kodezeilen führen.

## Mehrzeilige Strings

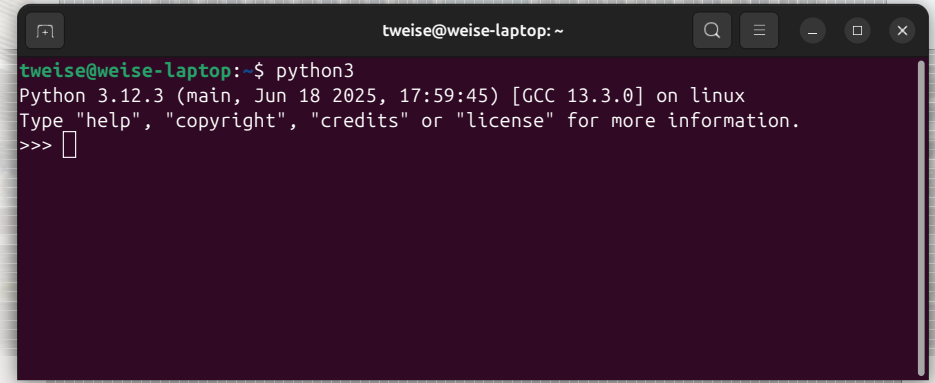



- Wir können Strings erstellen, die über mehrere Zeilen gehen, in dem wir die Escape-Sequenz `\n` einfügen.
- Wenn die Strings lang werden, dann kann das aber zu häßlichen sehr langen Kodezeilen führen.
- Für lange mehrzeilige Strings gibt es die Syntax mit dreifachen doppelten Anführungszeichen: `"""..."""`.

# Mehrzeilige Strings

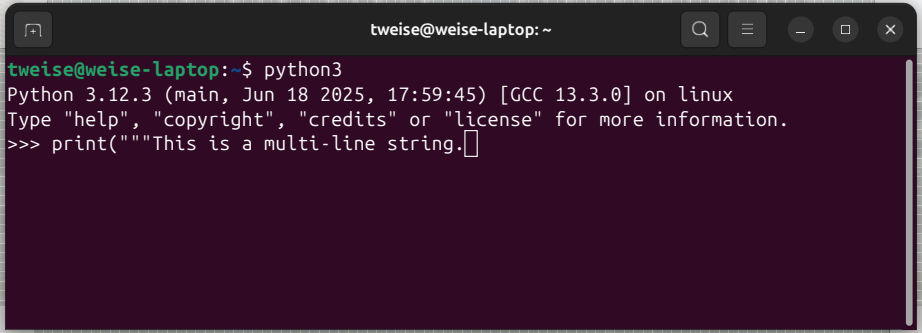


- Für lange mehrzeilige Strings gibt es die Syntax mit dreifachen doppelten Anführungszeichen: `"""..."""`.



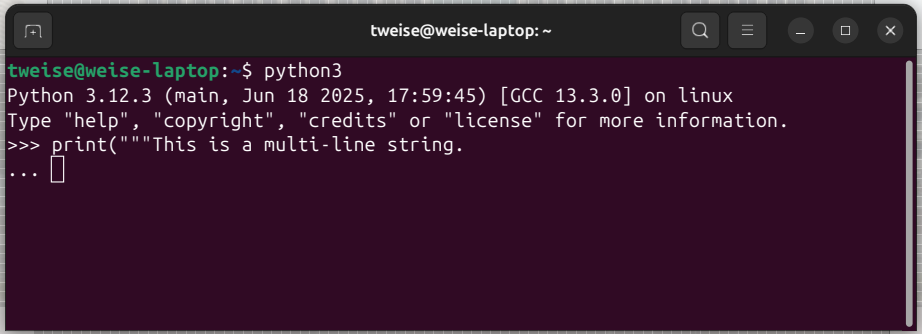
# Mehrzeilige Strings

- Wir beginnen solche Strings immer mit `"""` anstelle von `"`.



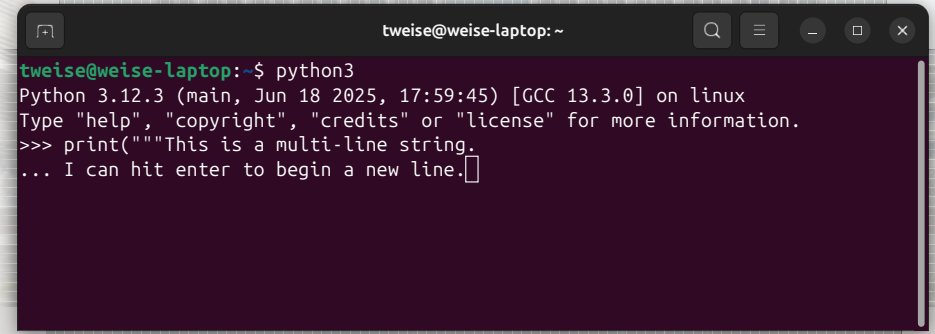
# Mehrzeilige Strings

- Wir beginnen solche Strings immer mit `"""` anstelle von `"`. Machen wir einen Zeilenumbruch in so einem String...



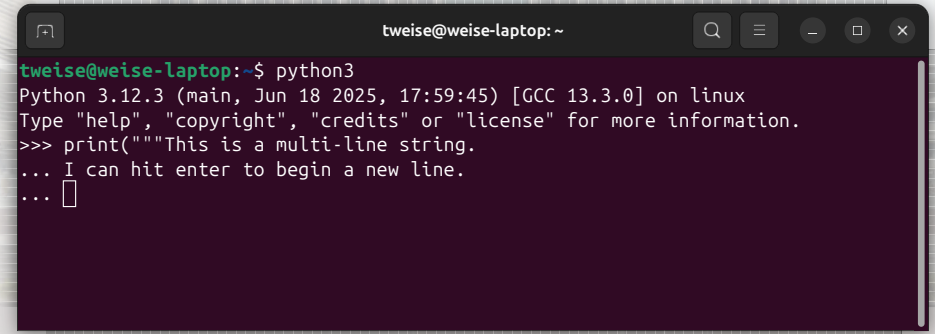
# Mehrzeilige Strings

- Wir beginnen solche Strings immer mit `"""` anstelle von `"`. Machen wir einen Zeilenumbruch in so einem String, dann können wir wieder auf der nächsten Zeile weiterschreiben. . .



# Mehrzeilige Strings

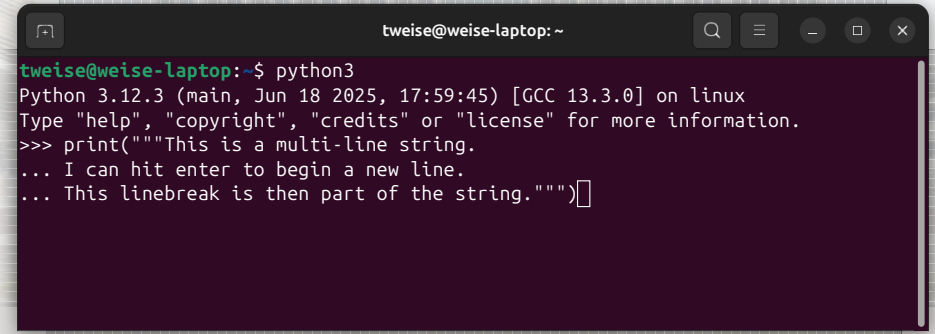
- Wir beginnen solche Strings immer mit `"""` anstelle von `"`. Machen wir einen Zeilenumbruch in so einem String, dann können wir wieder auf der nächsten Zeile weiterschreiben, die im Interpreter wieder mit drei Punkten anfängt.



# Mehrzeilige Strings

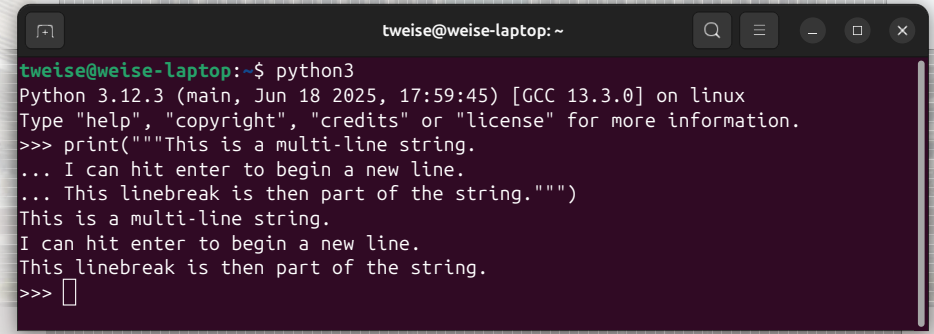


- Wir beginnen solche Strings immer mit `"""` anstelle von `"`. Machen wir einen Zeilenumbruch in so einem String, dann können wir wieder auf der nächsten Zeile weiterschreiben, die im Interpreter wieder mit drei Punkten anfängt. Wir können so beliebig viele Zeilenumbrüche schreiben.



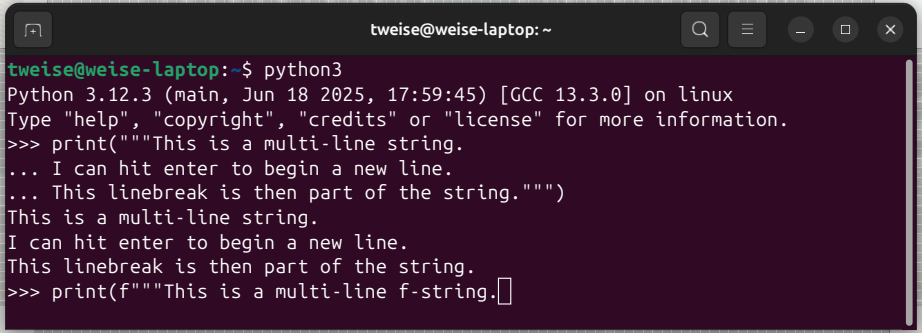
# Mehrzeilige Strings

- Wir beginnen solche Strings immer mit `"""` anstelle von `"`. Machen wir einen Zeilenumbruch in so einem String, dann können wir wieder auf der nächsten Zeile weiterschreiben, die im Interpreter wieder mit drei Punkten anfängt. Wir können so beliebig viele Zeilenumbrüche schreiben. Diese sind dann Teil des Strings (anders als bei `\` an Zeilenende!).



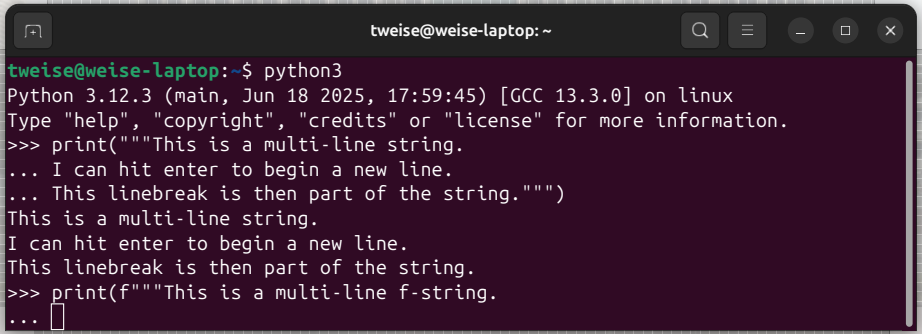
# Mehrzeilige Strings

- Diese mehrzeilige Syntax lässt sich mit f-Strings kombinieren.



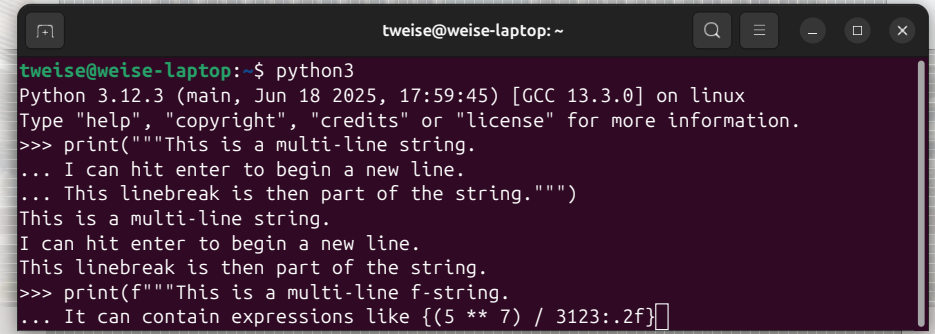
# Mehrzeilige Strings

- Diese mehrzeilige Syntax lässt sich mit f-Strings kombinieren. Solche mehrzeiligen f-Strings sind von `f"""..."""` begrenzt.



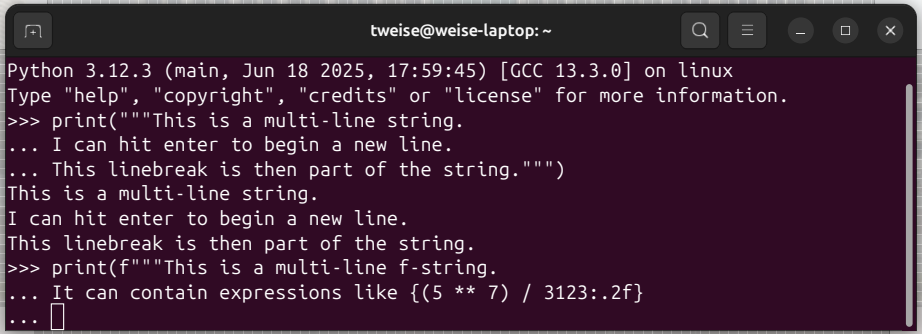
# Mehrzeilige Strings

- Diese mehrzeilige Syntax lässt sich mit f-Strings kombinieren. Solche mehrzeiligen f-Strings sind von `f"""..."""` begrenzt. Sie können beliebige Ausdrücke enthalten und normal interpoliert.



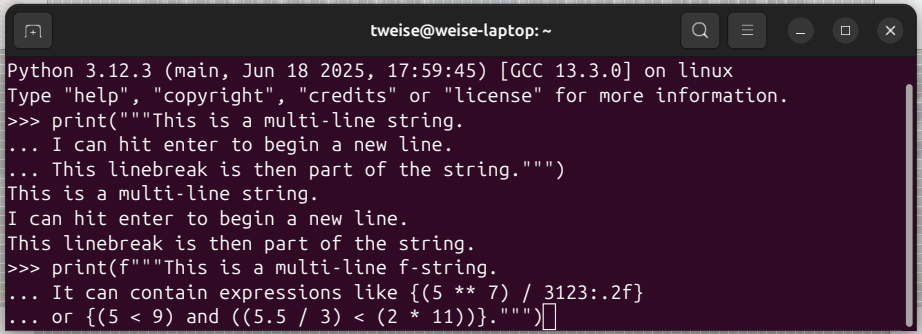
# Mehrzeilige Strings

- Diese mehrzeilige Syntax lässt sich mit f-Strings kombinieren. Solche mehrzeiligen f-Strings sind von `f"""..."""` begrenzt. Sie können beliebige Ausdrücke enthalten und normal interpoliert.



# Mehrzeilige Strings

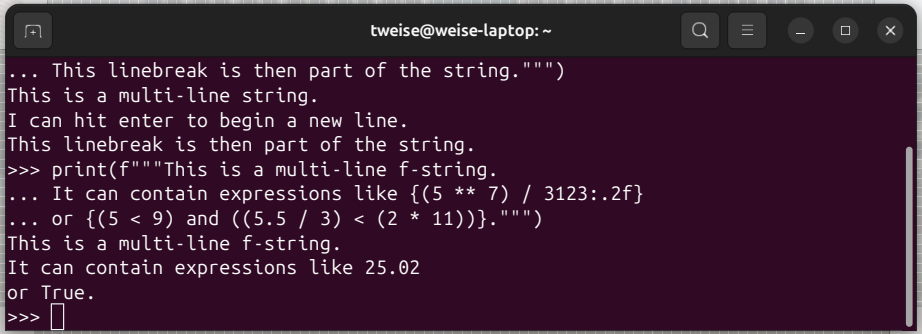
- Diese mehrzeilige Syntax lässt sich mit f-Strings kombinieren. Solche mehrzeiligen f-Strings sind von `f"""..."""` begrenzt. Sie können beliebige Ausdrücke enthalten und normal interpoliert.



# Mehrzeilige Strings



- Diese mehrzeilige Syntax lässt sich mit f-Strings kombinieren. Solche mehrzeiligen f-Strings sind von `f"""..."""` begrenzt. Sie können beliebige Ausdrücke enthalten und normal interpoliert.



## Mehrzeilige Strings




- Somit haben wir nun auch eine Methode, große Textstücke als Strings in unsere Programme einzufügen.

## Mehrzeilige Strings




- Somit haben wir nun auch eine Methode, große Textstücke als Strings in unsere Programme einzufügen.
- Das wird später bei der Dokumentation von Kode nützlich.

## Mehrzeilige Strings




- Somit haben wir nun auch eine Methode, große Textstücke als Strings in unsere Programme einzufügen.
- Das wird später bei der Dokumentation von Kode nützlich.

**Gute Praxis**

Wenn wir mehrzeilige Strings definieren, dann sollte immer doppelte Anführungszeichen verwendet werden und nicht einfache (natürlich immer in dreifacher Ausfertigung).[27,69]

# Unicode



# Texte als Zahlen

- Der Speicher unserer Computer speichert Gruppen von Bits in fester Größe, sagen wir, Bytes zu jeweils 8 Bit.

# Texte als Zahlen




- Der Speicher unserer Computer speichert Gruppen von Bits in fester Größe, sagen wir, Bytes zu jeweils 8 Bit.
- Diese Bytes werden oft als Ganzzahlen betrachtet.

# Texte als Zahlen




- Der Speicher unserer Computer speichert Gruppen von Bits in fester Größe, sagen wir, Bytes zu jeweils 8 Bit.
- Diese Bytes werden oft als Ganzzahlen betrachtet.
- Während Python „beliebig-große“ `ints` anbietet, die aus entsprechend vielen Bytes bestehen, nutzen wir in anderen Programmiersprachen oftmals jeweils 8 Bytes, um einen Integer darzustellen.

# Texte als Zahlen




- Der Speicher unserer Computer speichert Gruppen von Bits in fester Größe, sagen wir, Bytes zu jeweils 8 Bit.
- Diese Bytes werden oft als Ganzzahlen betrachtet.
- Während Python „beliebig-große“ `ints` anbietet, die aus entsprechend vielen Bytes bestehen, nutzen wir in anderen Programmiersprachen oftmals jeweils 8 Bytes, um einen Integer darzustellen.
- Der Datentyp `float` ist in Python ebenfalls 8 Bytes groß – wie diese interpretiert werden haben Sie schon gelernt.

# Texte als Zahlen




- Der Speicher unserer Computer speichert Gruppen von Bits in fester Größe, sagen wir, Bytes zu jeweils 8 Bit.
- Diese Bytes werden oft als Ganzzahlen betrachtet.
- Während Python „beliebig-große“ `ints` anbietet, die aus entsprechend vielen Bytes bestehen, nutzen wir in anderen Programmiersprachen oftmals jeweils 8 Bytes, um einen Integer darzustellen.
- Der Datentyp `float` ist in Python ebenfalls 8 Bytes groß – wie diese interpretiert werden haben Sie schon gelernt.
- Aber wie funktioniert das mit Text?

# Texte als Zahlen




- Der Speicher unserer Computer speichert Gruppen von Bits in fester Größe, sagen wir, Bytes zu jeweils 8 Bit.
- Diese Bytes werden oft als Ganzzahlen betrachtet.
- Während Python „beliebig-große“ `ints` anbietet, die aus entsprechend vielen Bytes bestehen, nutzen wir in anderen Programmiersprachen oftmals jeweils 8 Bytes, um einen Integer darzustellen.
- Der Datentyp `float` ist in Python ebenfalls 8 Bytes groß – wie diese interpretiert werden haben Sie schon gelernt.
- Aber wie funktioniert das mit Text?
- In dem jedes Zeichen eines Textes als eine Zahl dargestellt wird.

# Texte als Zahlen




- Der Speicher unserer Computer speichert Gruppen von Bits in fester Größe, sagen wir, Bytes zu jeweils 8 Bit.
- Diese Bytes werden oft als Ganzzahlen betrachtet.
- Während Python „beliebig-große“ `ints` anbietet, die aus entsprechend vielen Bytes bestehen, nutzen wir in anderen Programmiersprachen oftmals jeweils 8 Bytes, um einen Integer darzustellen.
- Der Datentyp `float` ist in Python ebenfalls 8 Bytes groß – wie diese interpretiert werden haben Sie schon gelernt.
- Aber wie funktioniert das mit Text?
- In dem jedes Zeichen eines Textes als eine Zahl dargestellt wird.
- Ein `str` ist dann nichts als eine Sequenz von Zahlen.

# Texte als Zahlen




- Der Speicher unserer Computer speichert Gruppen von Bits in fester Größe, sagen wir, Bytes zu jeweils 8 Bit.
- Diese Bytes werden oft als Ganzzahlen betrachtet.
- Während Python „beliebig-große“ `ints` anbietet, die aus entsprechend vielen Bytes bestehen, nutzen wir in anderen Programmiersprachen oftmals jeweils 8 Bytes, um einen Integer darzustellen.
- Der Datentyp `float` ist in Python ebenfalls 8 Bytes groß – wie diese interpretiert werden haben Sie schon gelernt.
- Aber wie funktioniert das mit Text?
- In dem jedes Zeichen eines Textes als eine Zahl dargestellt wird.
- Ein `str` ist dann nichts als eine Sequenz von Zahlen.
- Das System weiß, welches Zeichen zu welcher Zahl gehört.

# Zeichen als Zahlen




- Historisch gesehen gibt es viele „Zeichen-zu-Zahlen“ Mappings.

# Zeichen als Zahlen




- Historisch gesehen gibt es viele „Zeichen-zu-Zahlen“ Mappings.
- Die bekannteste solche Zuordnung ist ASCII[1,68], welche sieben Bits pro Zeichen verwendete.

# Zeichen als Zahlen




- Historisch gesehen gibt es viele „Zeichen-zu-Zahlen“ Mappings.
- Die bekannteste solche Zuordnung ist ASCII[1,68], welche sieben Bits pro Zeichen verwendete.
- Darum konnte ASCII auch nur Lateinische Buchstaben, Satzzeichen, Ziffern, und ein Kontrollzeichen (wie `\t` und `\n`...) darstellen.

# Zeichen als Zahlen




- Historisch gesehen gibt es viele „Zeichen-zu-Zahlen“ Mappings.
- Die bekannteste solche Zuordnung ist ASCII[1,68], welche sieben Bits pro Zeichen verwendete.
- Darum konnte ASCII auch nur Lateinische Buchstaben, Satzzeichen, Ziffern, und ein Kontrollzeichen (wie `\t` und `\n`...) darstellen.
- Ihnen ist vielleicht aufgefallen, dass verschiedene Sprachen verschiedene Zeichen verwenden ... und zwar mehr als insgesamt $2^7 = 128$.

# Zeichen als Zahlen




- Historisch gesehen gibt es viele „Zeichen-zu-Zahlen“ Mappings.
- Die bekannteste solche Zuordnung ist ASCII[1,68], welche sieben Bits pro Zeichen verwendete.
- Darum konnte ASCII auch nur Lateinische Buchstaben, Satzzeichen, Ziffern, und ein Kontrollzeichen (wie `\t` und `\n`...) darstellen.
- Ihnen ist vielleicht aufgefallen, dass verschiedene Sprachen verschiedene Zeichen verwenden ... und zwar mehr als insgesamt $2^7 = 128$.
- Deshalb gibt es verschiedene solche Zuordnungen.

# Zeichen als Zahlen




- Historisch gesehen gibt es viele „Zeichen-zu-Zahlen“ Mappings.
- Die bekannteste solche Zuordnung ist ASCII[1,68], welche sieben Bits pro Zeichen verwendete.
- Darum konnte ASCII auch nur Lateinische Buchstaben, Satzzeichen, Ziffern, und ein Kontrollzeichen (wie `\t` und `\n`...) darstellen.
- Ihnen ist vielleicht aufgefallen, dass verschiedene Sprachen verschiedene Zeichen verwenden ... und zwar mehr als insgesamt $2^7 = 128$.
- Deshalb gibt es verschiedene solche Zuordnungen.
- In China existieren verschiedene, auf Chinesische Zeichen spezialisierte, Zuordnungen, z. B. das historische GB 2312[81], GBK[83], und das neuere GB 18030[82].

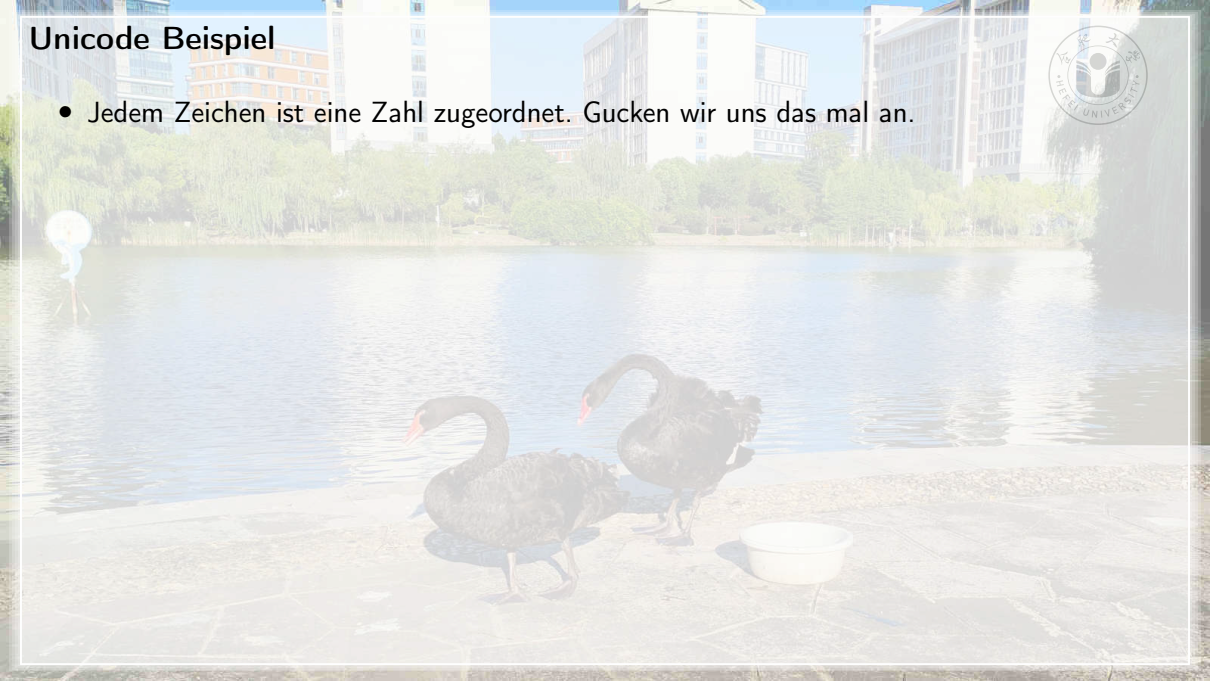
# Zeichen als Zahlen




- Historisch gesehen gibt es viele „Zeichen-zu-Zahlen“ Mappings.
- Die bekannteste solche Zuordnung ist ASCII[1,68], welche sieben Bits pro Zeichen verwendete.
- Darum konnte ASCII auch nur Lateinische Buchstaben, Satzzeichen, Ziffern, und ein Kontrollzeichen (wie `\t` und `\n`...) darstellen.
- Ihnen ist vielleicht aufgefallen, dass verschiedene Sprachen verschiedene Zeichen verwenden ... und zwar mehr als insgesamt $2^7 = 128$.
- Deshalb gibt es verschiedene solche Zuordnungen.
- In China existieren verschiedene, auf Chinesische Zeichen spezialisierte, Zuordnungen, z. B. das historische GB 2312[81], GBK[83], und das neuere GB 18030[82].
- Die große Mehrheit der Systeme heute (und Python) verwendet einen gemeinsamen Standard für alle Sprachen: Unicode[35,64,67].

## Unicode Beispiel

- Jedem Zeichen ist eine Zahl zugeordnet. Gucken wir uns das mal an.



# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | | 、 | 。 | 〃 | 〄 | 々 | 〆 | 〇 | 〈 | 〉 | 《 | 》 | 「 | 」 | 『 | 』 |
| 301 | 【 | 】 | 〒 | 〓 | 〔 | 〕 | 〖 | 〗 | 〘 | 〙 | 〚 | 〛 | 〜 | 〝 | 〞 | 〟 |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4f6 | 你 | 仦 | 佢 | 佣 | 佤 | 金 | 佑 | 佟 | 抱 | 佩 | 佪 | 佫 | 佬 | 佯 | 佮 | 佯 |
| 4f7 | 佰 | 佘 | 佫 | 佳 | 佴 | 併 | 佶 | 佷 | 佸 | 佹 | 佺 | 佻 | 佼 | 佽 | 佾 | 使 |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 597 | 奥 | 奠 | 轉 | 女 | 奴 | 奸 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 奾 | 奴 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妝 | 妇 | 妈 | 妩 | 妊 | 妖 | 妍 | 妍 | 妗 | 妓 |

# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | | 、 | 。 | 〃 | 〄 | 々 | 〆 | 〇 | 〈 | 〉 | 《 | 》 | 「 | 」 | 『 | 』 |
| 301 | 【 | 】 | 〒 | 〓 | 〔 | 〕 | 〖 | 〗 | 〘 | 〙 | 〚 | 〛 | 〜 | 〝 | 〞 | 〟 |

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4f6 | 你 | 价 | 佢 | 佣 | 佩 | 金 | 佑 | 作 | 包 | 佩 | 佃 | 佫 | 佬 | 佯 | 佮 | 佯 |
| 4f7 | 佰 | 金 | 佫 | 佳 | 佴 | 併 | 佶 | 很 | 佸 | 倦 | 佺 | 佻 | 佼 | 佽 | 佾 | 使 |

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 597 | 奥 | 奠 | 轉 | 女 | 奴 | 奶 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 奾 | 奴 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妝 | 妇 | 妈 | 妩 | 妊 | 妹 | 妍 | 妍 | 妗 | 妓 |

你

\u

# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | | 、 | 。 | 〃 | 〄 | 々 | 〆 | 〇 | 〈 | 〉 | 《 | 》 | 「 | 」 | 『 | 』 |
| 301 | 【 | 】 | 〒 | 〓 | 〔 | 〕 | 〖 | 〗 | 〘 | 〙 | 〚 | 〛 | 〜 | 〝 | 〞 | 〟 |

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4f6 | 你 | 价 | 佢 | 佣 | 佩 | 金 | 佑 | 作 | 佗 | 佩 | 佃 | 佫 | 佬 | 佯 | 佮 | 佯 |
| 4f7 | 佰 | 佘 | 佫 | 佳 | 佴 | 併 | 佶 | 很 | 佸 | 佗 | 佺 | 佻 | 佼 | 佽 | 佾 | 使 |

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 597 | 奥 | 奠 | 轉 | 女 | 奴 | 奸 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 奾 | 奴 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妝 | 妇 | 妈 | 妩 | 妊 | 妖 | 妍 | 妍 | 妗 | 妓 |

你

\u

# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | | 、 | 。 | 〃 | 〄 | 々 | 〆 | 〇 | 〈 | 〉 | 《 | 》 | 「 | 」 | 『 | 』 |
| 301 | 【 | 】 | 〒 | 〓 | 〔 | 〕 | 〖 | 〗 | 〘 | 〙 | 〚 | 〛 | 〜 | 〝 | 〞 | 〟 |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4f6 | 你 | 仫 | 佢 | 佣 | 佤 | 金 | 佑 | 佟 | 包 | 佩 | 佃 | 佫 | 佬 | 佯 | 佮 | 佯 |
| 4f7 | 佰 | 佘 | 佫 | 佳 | 佴 | 併 | 佶 | 佷 | 佸 | 佗 | 佺 | 佻 | 佼 | 佽 | 佾 | 使 |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 597 | 奥 | 奠 | 轉 | 女 | 奴 | 奸 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 奾 | 奴 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妝 | 妇 | 妈 | 妩 | 妊 | 妖 | 妍 | 妍 | 妗 | 妓 |

你

\u

# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | | 、 | 。 | 〃 | 〄 | 々 | 〆 | 〇 | 〈 | 〉 | 《 | 》 | 「 | 」 | 『 | 』 |
| 301 | 【 | 】 | 〒 | 〓 | 〔 | 〕 | 〖 | 〗 | 〘 | 〙 | 〚 | 〛 | 〜 | 〝 | 〞 | 〟 |

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4f6 | 你 | 仦 | 佢 | 佣 | 佤 | 金 | 佑 | 佟 | 包 | 佩 | 佃 | 佫 | 佬 | 佯 | 佮 | 佯 |
| 4f7 | 佰 | 佘 | 佫 | 佳 | 佴 | 併 | 佶 | 佷 | 佸 | 佗 | 佺 | 佻 | 佼 | 佽 | 佾 | 使 |

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 597 | 奥 | 奠 | 轉 | 女 | 奴 | 奸 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 奾 | 奴 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妝 | 妇 | 妈 | 妩 | 妊 | 妓 | 妍 | 妍 | 妗 | 妓 |

你

\u4f6

# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | ␣ | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |
| 300 | ␣ | 、 | 。 | 〃 | 〄 | 々 | 〆 | 〇 | 〈 | 〉 | 《 | 》 | 「 | 」 | 『 | 』 |
| 301 | 【 | 】 | 〒 | 〓 | 〔 | 〕 | 〖 | 〗 | 〘 | 〙 | 〚 | 〛 | 〜 | 〝 | 〞 | 〟 |
| 4f6 | 你 | 仦 | 佢 | 佣 | 佤 | 金 | 佑 | 佟 | 抱 | 佩 | 佃 | 佫 | 佬 | 佯 | 佮 | 佯 |
| 4f7 | 佰 | 佘 | 佫 | 佳 | 佴 | 併 | 佶 | 佷 | 佸 | 佗 | 佺 | 佻 | 佼 | 佽 | 佾 | 使 |
| 597 | 奥 | 奠 | 轟 | 女 | 奴 | 奸 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 奾 | 奴 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妝 | 妇 | 妈 | 妩 | 妊 | 妹 | 妍 | 妍 | 妗 | 妓 |

你

\u4f6

# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | ␣ | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |
| 300 | ␣ | 、 | 。 | 〃 | 〄 | 々 | 〆 | 〇 | 〈 | 〉 | 《 | 》 | 「 | 」 | 『 | 』 |
| 301 | 【 | 】 | 〒 | 〓 | 〔 | 〕 | 〖 | 〗 | 〘 | 〙 | 〚 | 〛 | 〜 | 〝 | 〞 | 〟 |
| 4f6 | 你 | 仦 | 佢 | 佣 | 佤 | 金 | 佑 | 伟 | 包 | 佩 | 佃 | 佫 | 佬 | 佛 | 佮 | 佯 |
| 4f7 | 佰 | 佘 | 佫 | 佳 | 佴 | 併 | 佶 | 佷 | 佸 | 佹 | 佺 | 佻 | 佼 | 佽 | 佾 | 使 |
| 597 | 奥 | 奠 | 轟 | 女 | 奴 | 奸 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 奾 | 奴 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妝 | 妇 | 妈 | 妩 | 妊 | 妹 | 妍 | 妍 | 妗 | 妓 |

你

\u4f60

# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | ␣ | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |
| 300 | ␣ | 、 | 。 | 〃 | 〄 | 々 | 〆 | 〇 | 〈 | 〉 | 《 | 》 | 「 | 」 | 『 | 』 |
| 301 | 【 | 】 | 〒 | 〓 | 〔 | 〕 | 〖 | 〗 | 〘 | 〙 | 〚 | 〛 | 〜 | 〝 | 〞 | 〟 |
| 4f6 | 你 | 仦 | 佢 | 佣 | 佤 | 金 | 佑 | 伟 | 佨 | 佩 | 佪 | 佫 | 佬 | 佯 | 佮 | 佯 |
| 4f7 | 佰 | 佘 | 佫 | 佳 | 佴 | 併 | 佶 | 佷 | 佸 | 佹 | 佺 | 佻 | 佼 | 佽 | 佾 | 使 |
| 597 | 奥 | 奠 | 轟 | 女 | 奴 | 奸 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 奾 | 奴 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妝 | 妇 | 妈 | 妩 | 妊 | 妖 | 妍 | 妍 | 妗 | 妓 |

你 好

\u4f60 \u

# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | _ | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |
| 300 | _ | 、 | 。 | 〃 | 〄 | 々 | 〆 | 〇 | 〈 | 〉 | 《 | 》 | 「 | 」 | 『 | 』 |
| 301 | 【 | 】 | 〒 | 〓 | 〔 | 〕 | 〖 | 〗 | 〘 | 〙 | 〚 | 〛 | 〜 | 〝 | 〞 | 〟 |
| 4f6 | 你 | 仦 | 佢 | 佣 | 佤 | 金 | 佑 | 伟 | 包 | 佩 | 佃 | 佫 | 佬 | 侰 | 佮 | 佯 |
| 4f7 | 佰 | 佘 | 佫 | 佳 | 佴 | 併 | 佶 | 佷 | 佸 | 佗 | 佺 | 佻 | 佼 | 佽 | 佾 | 使 |
| 597 | 奥 | 奠 | 轉 | 女 | 奴 | 奸 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 奾 | 奴 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妝 | 妇 | 妈 | 妩 | 妊 | 妹 | 妍 | 妍 | 妗 | 妓 |

你 好

\u4f60 \u

# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | ␣ | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |
| 300 | ␣ | 、 | 。 | 〃 | 〄 | 々 | 〆 | 〇 | 〈 | 〉 | 《 | 》 | 「 | 」 | 『 | 』 |
| 301 | 【 | 】 | 〒 | 〓 | 〔 | 〕 | 〖 | 〗 | 〘 | 〙 | 〚 | 〛 | 〜 | 〝 | 〞 | 〟 |
| 4f6 | 你 | 伀 | 佢 | 佣 | 佤 | 金 | 佑 | 作 | 佗 | 佩 | 佪 | 佫 | 佬 | 佯 | 佮 | 佯 |
| 4f7 | 佰 | 金 | 佫 | 佳 | 佴 | 併 | 佶 | 佷 | 佸 | 佹 | 佺 | 佻 | 佼 | 佽 | 佾 | 使 |
| 597 | 奥 | 奠 | 奭 | 女 | 奴 | 奶 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 妣 | 妣 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妝 | 妇 | 妈 | 妩 | 妊 | 妹 | 妍 | 妍 | 妗 | 妓 |

你 好

\u4f60 \u

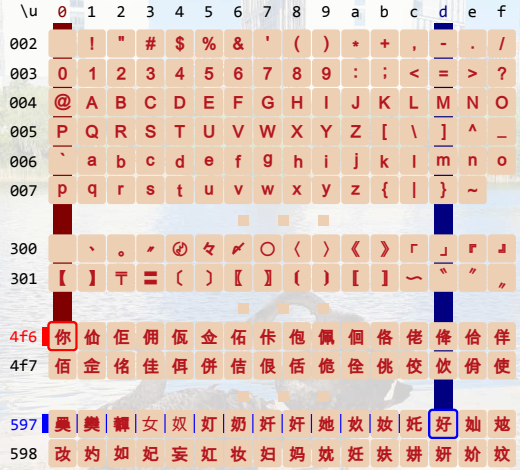
# Unicode Beispiel




你 好

\u4f60 \u597

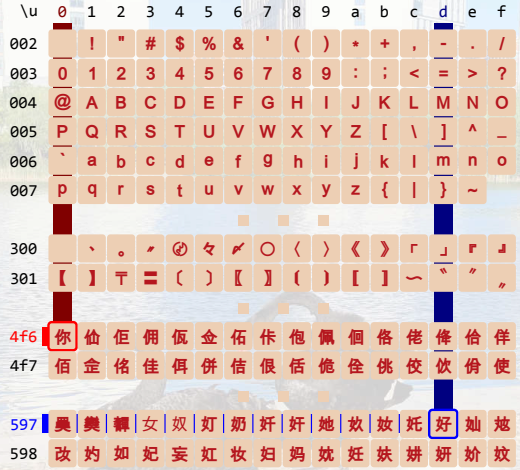
# Unicode Beispiel




你 好

\u4f60 \u597

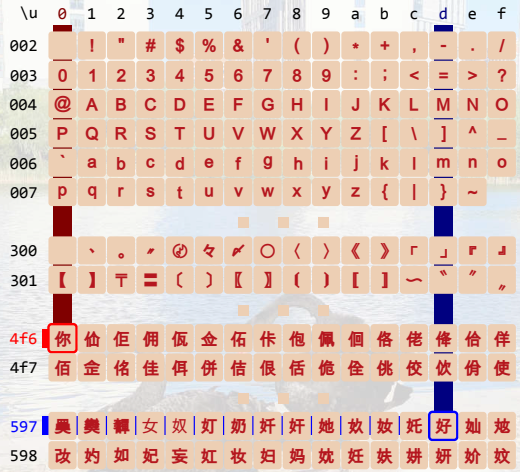
# Unicode Beispiel




你 好

\u4f60 \u597d

# Unicode Beispiel




你 好 。

\u4f60 \u597d \u

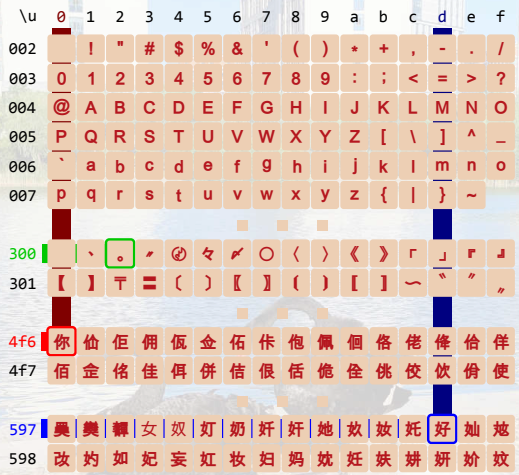
# Unicode Beispiel




你 好 。

\u4f60 \u597d \u

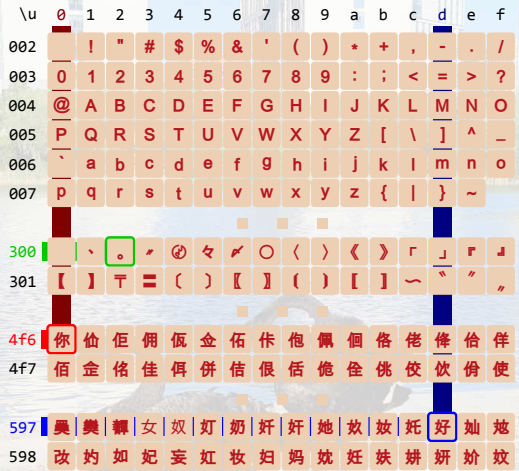
# Unicode Beispiel




你 好 。

\u4f60 \u597d \u

# Unicode Beispiel




你 好 。

\u4f60 \u597d \u300

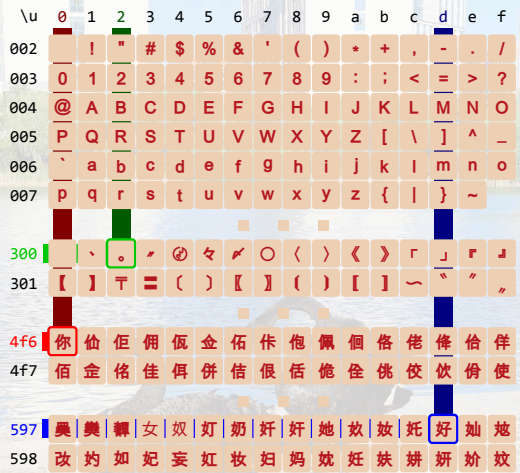
# Unicode Beispiel

| \u | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 002 | ␣ | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 003 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 004 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 005 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 006 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 007 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |
| 300 | ◌̀ | ◌́ | ◌̂ | ◌̃ | ◌̄ | ◌̆ | ◌̇ | ◌̈ | ◌̉ | ◌̊ | ◌̋ | ◌̌ | ◌̍ | ◌̎ | ◌̏ | ◌̐ |
| 301 | ◌̑ | ◌̒ | ◌̓ | ◌̔ | ◌̕ | ◌̖ | ◌̗ | ◌̘ | ◌̙ | ◌̚ | ◌̛ | ◌̜ | ◌̝ | ◌̞ | ◌̟ | ◌̠ |
| 4f6 | 你 | 价 | 佢 | 佣 | 佩 | 金 | 佑 | 作 | 抱 | 佩 | 佃 | 格 | 佬 | 修 | 俭 | 伴 |
| 4f7 | 佰 | 金 | 络 | 佳 | 俱 | 併 | 信 | 很 | 恬 | 俺 | 佺 | 佻 | 佼 | 伙 | 俘 | 使 |
| 597 | 奥 | 奠 | 嬋 | 女 | 奴 | 奶 | 奶 | 奸 | 奸 | 她 | 奴 | 奴 | 奼 | 好 | 妯 | 她 |
| 598 | 改 | 妁 | 如 | 妃 | 妄 | 妊 | 妆 | 妇 | 妈 | 妩 | 妊 | 妖 | 妍 | 妍 | 妗 | 妓 |




\u4f60 \u597d \u300

# Unicode Beispiel

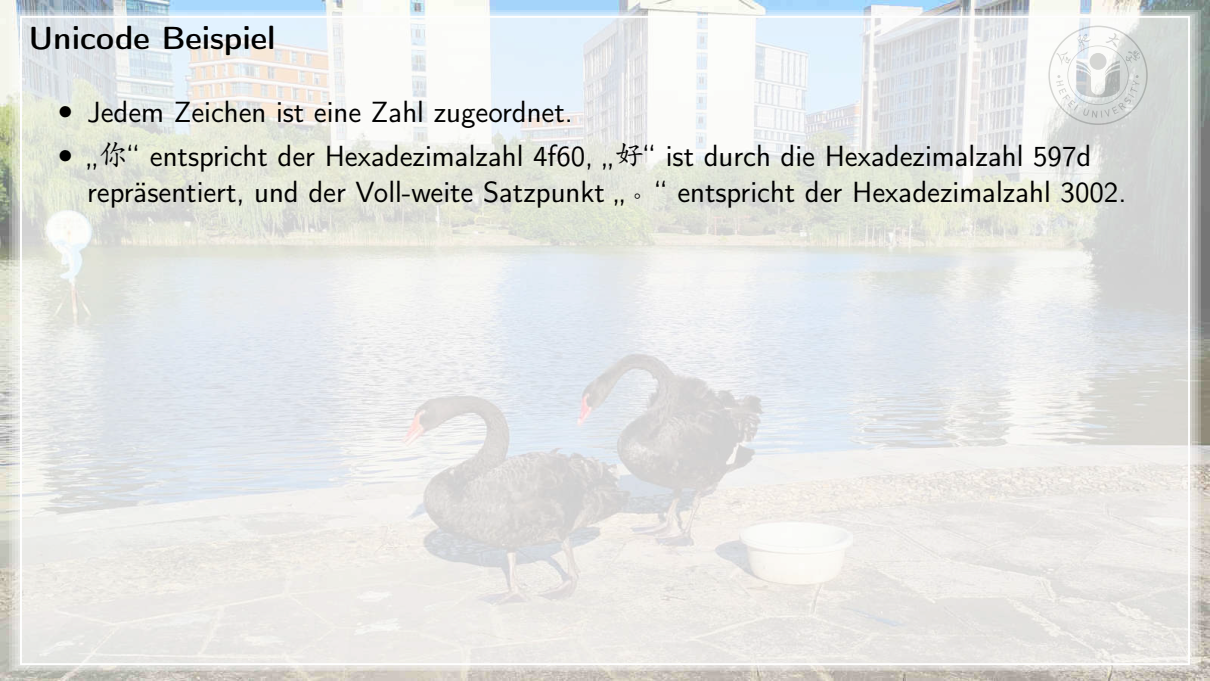



你 好 。

\u4f60 \u597d \u3002

## Unicode Beispiel

- Jedem Zeichen ist eine Zahl zugeordnet.
- „你“ entspricht der Hexadezimalzahl 4f60, „好“ ist durch die Hexadezimalzahl 597d repräsentiert, und der Voll-weite Satzpunkt „。“ entspricht der Hexadezimalzahl 3002.



## Unicode Beispiel

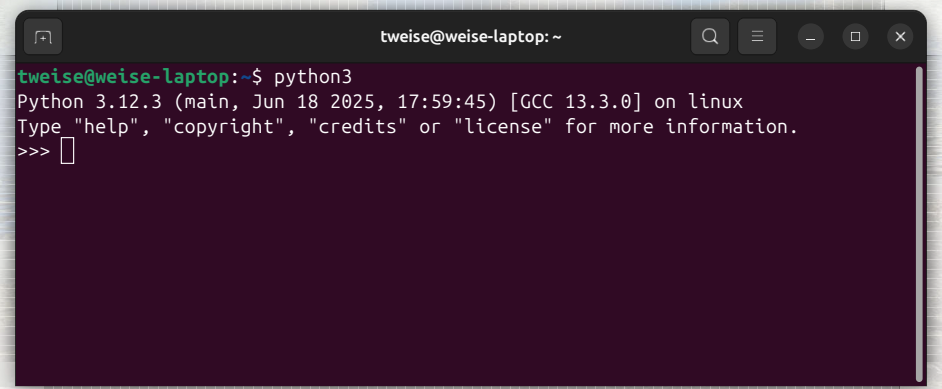
- Jedem Zeichen ist eine Zahl zugeordnet.
- „你“ entspricht der Hexadezimalzahl 4f60, „好“ ist durch die Hexadezimalzahl 597d repräsentiert, und der Voll-weite Satzpunkt „。“ entspricht der Hexadezimalzahl 3002.
- Alle Zeichen können als so-genanntes Unicode-Escape, also `\u` gefolgt von der entsprechenden Hexadezimalzahl geschrieben werden.

## Unicode Beispiel

- Jedem Zeichen ist eine Zahl zugeordnet.
- „你“ entspricht der Hexadezimalzahl 4f60, „好“ ist durch die Hexadezimalzahl 597d repräsentiert, und der Voll-weite Satzpunkt „。“ entspricht der Hexadezimalzahl 3002.
- Alle Zeichen können als so-genanntes Unicode-Escape, also `\u` gefolgt von der entsprechenden Hexadezimalzahl geschrieben werden.
- Gucken wir uns das mal an.

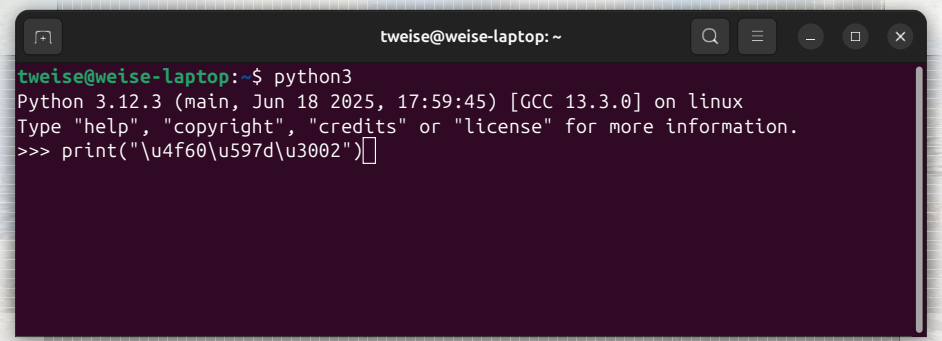
# Unicode Beispiel

- Gucken wir uns das mal an.



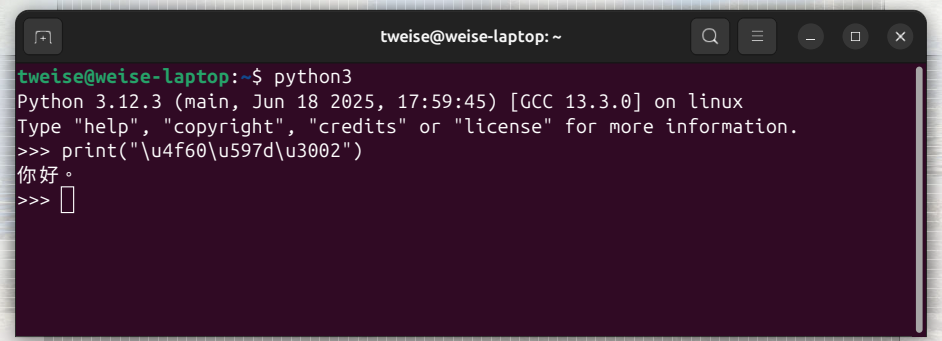
# Unicode Beispiel

- Wir verwenden die Unicode-Escapes `\u4f60`, `\u597d`, und `\u3002` für die drei Zeichen „你好。“.



# Unicode Beispiel

- Wir verwenden die Unicode-Escapes `\u4f60`, `\u597d`, und `\u3002` für die drei Zeichen „你好。“. Und tatsächlich wird der korrekte Text gedruckt.



## Unicode Beispiel

- Jedem Zeichen ist eine Zahl zugeordnet.
- „你“ entspricht der Hexadezimalzahl 4f60, „好“ ist durch die Hexadezimalzahl 597d repräsentiert, und der Voll-weite Satzpunkt „。“ entspricht der Hexadezimalzahl 3002.
- Alle Zeichen können als so-genanntes Unicode-Escape, also `\u` gefolgt von der entsprechenden Hexadezimalzahl geschrieben werden.
- Wir verwenden die Unicode-Escapes `\u4f60`, `\u597d`, und `\u3002` für die drei Zeichen „你好。“. Und tatsächlich wird der korrekte Text gedruckt.
- Damit haben Sie nun also auch eine Vorstellung, wie Text im Speicher des Computers dargstellt wird.

# Zusammenfassung

## Zusammenfassung




- Zeichenketten, Strings, also Instanzen der Klasse `str` sind wichtig.

## Zusammenfassung



- Zeichenketten, Strings, also Instanzen der Klasse `str` sind wichtig.
- Oftmals bekommen wir Daten in Form von solchen Zeichenketten als Input für unsere Programme.

## Zusammenfassung




- Zeichenketten, Strings, also Instanzen der Klasse `str` sind wichtig.
- Oftmals bekommen wir Daten in Form von solchen Zeichenketten als Input für unsere Programme.
- Wir wandeln sie dann in andere Datentypen, with `int` oder `float` um, um mit ihnen zu arbeiten.

## Zusammenfassung




- Zeichenketten, Strings, also Instanzen der Klasse `str` sind wichtig.
- Oftmals bekommen wir Daten in Form von solchen Zeichenketten als Input für unsere Programme.
- Wir wandeln sie dann in andere Datentypen, with `int` oder `float` um, um mit ihnen zu arbeiten.
- Danach geben wir die Ergebnisse unserer Berechnungen wieder als Text aus. Dabei können wir z. B. f-Strings nutzen, um andere Datentypen elegant als Strings zu formatieren.

## Zusammenfassung



- Zeichenketten, Strings, also Instanzen der Klasse `str` sind wichtig.
- Oftmals bekommen wir Daten in Form von solchen Zeichenketten als Input für unsere Programme.
- Wir wandeln sie dann in andere Datentypen, with `int` oder `float` um, um mit ihnen zu arbeiten.
- Danach geben wir die Ergebnisse unserer Berechnungen wieder als Text aus. Dabei können wir z. B. f-Strings nutzen, um andere Datentypen elegant als Strings zu formatieren.
- Manchmal wollen wir auch mit Strings direkt arbeiten, z. B. Teile von ihnen extrahieren oder Groß- in Kleinbuchstaben umwandeln.

## Zusammenfassung



- Zeichenketten, Strings, also Instanzen der Klasse `str` sind wichtig.
- Oftmals bekommen wir Daten in Form von solchen Zeichenketten als Input für unsere Programme.
- Wir wandeln sie dann in andere Datentypen, with `int` oder `float` um, um mit ihnen zu arbeiten.
- Danach geben wir die Ergebnisse unserer Berechnungen wieder als Text aus. Dabei können wir z. B. f-Strings nutzen, um andere Datentypen elegant als Strings zu formatieren.
- Manchmal wollen wir auch mit Strings direkt arbeiten, z. B. Teile von ihnen extrahieren oder Groß- in Kleinbuchstaben umwandeln.
- Wir können Sonderzeichen durch Escape-Sequenzen in Strings einführen und mehrzeilige Strings erstellen.

## Zusammenfassung



- Zeichenketten, Strings, also Instanzen der Klasse `str` sind wichtig.
- Oftmals bekommen wir Daten in Form von solchen Zeichenketten als Input für unsere Programme.
- Wir wandeln sie dann in andere Datentypen, with `int` oder `float` um, um mit ihnen zu arbeiten.
- Danach geben wir die Ergebnisse unserer Berechnungen wieder als Text aus. Dabei können wir z. B. f-Strings nutzen, um andere Datentypen elegant als Strings zu formatieren.
- Manchmal wollen wir auch mit Strings direkt arbeiten, z. B. Teile von ihnen extrahieren oder Groß- in Kleinbuchstaben umwandeln.
- Wir können Sonderzeichen durch Escape-Sequenzen in Strings einführen und mehrzeilige Strings erstellen.
- Intern werden die Zeichen einer Zeichenkette durch Zahlen repräsentiert, wobei die Zuordnung von Zahlen zu Zeichen meist auf Unicode basiert.

## Zusammenfassung




- Zeichenketten, Strings, also Instanzen der Klasse `str` sind wichtig.
- Oftmals bekommen wir Daten in Form von solchen Zeichenketten als Input für unsere Programme.
- Wir wandeln sie dann in andere Datentypen, with `int` oder `float` um, um mit ihnen zu arbeiten.
- Danach geben wir die Ergebnisse unserer Berechnungen wieder als Text aus. Dabei können wir z. B. f-Strings nutzen, um andere Datentypen elegant als Strings zu formatieren.
- Manchmal wollen wir auch mit Strings direkt arbeiten, z. B. Teile von ihnen extrahieren oder Groß- in Kleinbuchstaben umwandeln.
- Wir können Sonderzeichen durch Escape-Sequenzen in Strings einführen und mehrzeilige Strings erstellen.
- Intern werden die Zeichen einer Zeichenkette durch Zahlen repräsentiert, wobei die Zuordnung von Zahlen zu Zeichen meist auf Unicode basiert.
- Das war schon recht viel. Hat aber auch Spaß gemacht.

谢谢您们！

Thank you!

Vielen Dank!



# References I


[1] *American Standard Code for Information Interchange (ASCII 1963)*. Techn. Ber. ASA X3.4-1963. New York, NY, USA: American Standards AssociationIncorporated, 17. Juni 1963. URL: https://www.sensitiveresearch.com/Archive/CharCodeHist/Files/CODES%20standards%20documents%20ASCII%20Sean%20Leonard%20Oct%202015/ASCII%2063,%20X3.4-1963.pdf (besucht am 2024-07-26) (siehe S. 277–283).

[2] 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. 332).

[3] Richard Beigel. "Irrationality without Number Theory". *The American Mathematical Monthly* 98(4):332–335, Apr. 1991. London, England, UK: Taylor and Francis Ltd. ISSN: 1930-0972. doi:10.1080/00029890.1991.12000762. URL: https://cis.temple.edu/~beigel/papers/irrationality-amm.PS.gz (besucht am 2026-11-04) (siehe S. 331).

[4] Joshua Bloch. *Effective Java*. Reading, MA, USA: Addison-Wesley Professional, Mai 2008. ISBN: 978-0-321-35668-0 (siehe S. 332).

[5] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 332).

[6] Florian Bruhin. *Python f-Strings*. Winterthur, Switzerland: Bruhin Software, 31. Mai 2023. URL: https://fstring.help (besucht am 2024-07-25) (siehe S. 128–178).

[7] "Built-in Functions: class bool". In: *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library/functions.html#bool (besucht am 2025-10-10) (siehe S. 180–209).

[8] Stephan C. Carlson und The Editors of Encyclopaedia Britannica. *Golden Ratio*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 21. Okt. 2024. URL: https://www.britannica.com/science/golden-ratio (besucht am 2024-12-14) (siehe S. 333).

[9] 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. 331).

# References II


[10] 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. 332).

[11] 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. 333).

[12] 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. 332).

[13] 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. 333).

[14] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 332).

[15] 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/ontstu_a2010n10/ontstu_a2010n10p21.pdf (besucht am 2026-11-03). Also discusses LIU Hui's (刘徽) method of approximating $\pi$ (siehe S. 333).

[16] 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. 332).

[17] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 332).

[18] Slobodan Dmitrović. *Modern C for Absolute Beginners: A Friendly Introduction to the C Programming Language*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0224-9 (siehe S. 331).

# References III


[19] "Escape Sequences". In: *Python 3 Documentation. The Python Language Reference*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. Kap. 2.4.1.1. URL: https://docs.python.org/3/reference/lexical_analysis.html#escape-sequences (besucht am 2025-08-05) (siehe S. 218–226, 247).

[20] Euclid of Alexandria (Εὐκλείδης). *Euclid's Elements of Geometry (Στοιχεῖα). The Greek Text of J.L. Heiberg (1883-1885) from Euclidis Elementa, Edidit et Latine Interpretatus est I.L. Heiberg in Aedibus B. G. Teubneri, 1883-1885. Edited, and provided with a modern English translation, by Richard Fitzpatrick*. Hrsg. von Richard Fitzpatrick. Übers. von Johan Ludvig Heiberg. revised and corrected. Austin, TX, USA: The University of Texas at Austin, 2008. ISBN: 978-0-615-17984-1. URL: https://farside.ph.utexas.edu/Books/Euclid/Elements.pdf (besucht am 2024-09-30) (siehe S. 333).

[21] Leonhard Euler. "An Essay on Continued Fractions". Übers. von Myra F. Wyman und Bostwick F. Wyman. *Mathematical Systems Theory* 18(1):295–328, Dez. 1985. New York, NY, USA: Springer Science+Business Media, LLC. ISSN: 1432-4350. doi:10.1007/BF01699475. URL: https://www.researchgate.net/publication/301720080 (besucht am 2024-09-24). Translation of [22]. (Siehe S. 323).

[22] Leonhard Euler. "De Fractionibus Continuis Dissertation". *Commentarii Academiae Scientiarum Petropolitanae* 9:98–137, 1737–1744. Petropolis (St. Petersburg), Russia: Typis Academiae. URL: https://scholarlycommons.pacific.edu/cgi/viewcontent.cgi?article=1070 (besucht am 2024-09-24). See [21] for a translation. (Siehe S. 323, 333).

[23] 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. 332).

[24] Michael Filaseta. "The Transcendence of $e$ and $\pi$". 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. 333).

[25] "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. 128–178).

# References IV


[26] 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. 128–137, 177, 178).

[27] David Goodger und Guido van Rossum. *Docstring Conventions*. Python Enhancement Proposal (PEP) 257. Beaverton, OR, USA: Python Software Foundation (PSF), 29. Mai–13. Juni 2001. URL: https://peps.python.org/pep-0257 (besucht am 2024-07-27) (siehe S. 22, 249–267).

[28] 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. 128–137, 177, 178).

[29] 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. 332).

[30] 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. 332).

[31] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 332).

[32] 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. 333).

[33] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISSN: 1863-7310. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 332).

# References V


[34] *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 (besucht am 2025-01-08). Consists of several parts, see https://modern-sql.com/standard for information where to obtain them. (Siehe S. 332).

[35] *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. 277–283, 333).

[36] Arthur Jones, Kenneth R. Pearson und Sidney A. Morris. "Transcendence of $e$ and $\pi$". 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. 333).

[37] 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. Translation of the historical Chinese mathematics book *Jiu Zhang Suanshu* (九章算术), including the commentaries of LIU Hui (刘徽). (Siehe S. 333).

[38] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISSN: 1863-7310. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 332).

[39] Marc Loy, Patrick Niemeyer und Daniel Leuck. *Learning Java*. 5. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2020. ISBN: 978-1-4920-5627-0 (siehe S. 332).

[40] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 332).

[41] "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. 333).

[42] 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. 128–137, 177, 178).

# References VI


[43] 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. ISSN: 1046-1698. ISBN: 978-1-55860-456-8 (siehe S. 332).

[44] Ivan Niven. "The Transcendence of $\pi$". *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. 333).

[45] 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). Discusses LIU Hui's (刘徽) method of approximating $\pi$ (siehe S. 333).

[46] 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. 332).

[47] *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).

[48] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 332).

[49] *Programming Languages – C, Working Document of SC22/WG14*. International Standard ISO/IEC9899:2017 C17 Ballot N2176. Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Nov. 2017. URL: https://files.lhmouse.com/standards/ISO%20C%20N2176.pdf (besucht am 2024-06-29) (siehe S. 331).

[50] 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. 331).

[51] 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. 332).

# References VII


[52] Neil James Alexander Sloane. *Decimal Expansion of Golden Ratio phi (or tau) = (1 + sqrt(5))/2*. Hrsg. von John Horton Conway. Bd. A001622 der Reihe The On-Line Encyclopedia of Integer Sequences. Highland Park, NJ, USA: The OEIS Foundation Inc., 13. Dez. 2024. URL: https://oeis.org/A001622 (besucht am 2024-12-14) (siehe S. 333).

[53] 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. 128–178).

[54] 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. 332).

[55] “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. 332).

[56] Ryan K. Stephens und Ronald R. Plew. *Sams Teach Yourself SQL in 21 Days*. 4. Aufl. Sams Teach Yourself. Indianapolis, IN, USA: SAMS Technical Publishing und Hoboken, NJ, USA: Pearson Education, Inc., Okt. 2002. ISBN: 978-0-672-32451-2 (siehe S. 327, 332).

[57] 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 of [56] (siehe S. 332).

[58] 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. 333).

[59] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 332).

[60] “Text Sequence Type – `str`”. In: *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library/stdtypes.html#textseq (besucht am 2024-07-25) (siehe S. 5–9).

# References VIII


[61] Alkin Tezuysal 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. 332).

[62] The Editors of Encyclopaedia Britannica, Hrsg. *Encyclopaedia Britannica*. Chicago, IL, USA: Encyclopædia Britannica, Inc.

[63] *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library (besucht am 2025-04-27).

[64] *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. 277–283, 333).

[65] 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. 332).

[66] "Truth Value Testing". In: *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library/stdtypes.html#truth (besucht am 2025-10-10) (siehe S. 180–209).

[67] *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. 277–283, 333).

[68] *USA Standard Code for Information Interchange (ASCII 1967)*. Techn. Ber. USAS X3.4-1967. New York, NY, USA: United States of America Standards Institute (USAS), 7. Juli 1967. URL: https://www.sensitiveresearch.com/Archive/CharCodeHist/Files/CODES%20standards%20documents%20ASCII%20Sean%20Leonard%20Oct%202015/ASCII%2068,%20X3.4-1967.pdf (besucht am 2024-07-26) (siehe S. 277–283).

[69] 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. 22, 249–267).

[70] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 332).

# References IX


[71] 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), 2025. URL: https://thomasweise.github.io/databases (besucht am 2025-01-05) (siehe S. 331, 332).

[72] Thomas Weise (汤卫思). *Programming with C*. 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), 2026. URL: https://thomasweise.github.io/programmingWithC (besucht am 2026-05-10) (siehe S. 331).

[73] 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), 2024–2026. URL: https://thomasweise.github.io/programmingWithPython (besucht am 2025-01-05) (siehe S. 332).

[74] Eric Wolfgang Weisstein. "Irrational Number". In: *MathWorld – A Wolfram Web Resource*. Champaign, IL, USA: Wolfram Research, Inc., 22. Aug. 2024. URL: https://mathworld.wolfram.com/IrrationalNumber.html (besucht am 2026-11-04) (siehe S. 331).

[75] Eric Wolfgang Weisstein. *MathWorld – A Wolfram Web Resource*. Champaign, IL, USA: Wolfram Research, Inc., 22. Aug. 2024. URL: https://mathworld.wolfram.com (besucht am 2024-09-24).

[76] Eric Wolfgang Weisstein. "Transcendental Number". In: *MathWorld – A Wolfram Web Resource*. Champaign, IL, USA: Wolfram Research, Inc., 22. Aug. 2024. URL: https://mathworld.wolfram.com/TranscendentalNumber.html (besucht am 2026-11-04) (siehe S. 332).

[77] *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. 332).

[78] 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. 331).

# References X


[79] 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. 333).

[80] 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. 331).

[81] 信息交换用汉字编码字符集 *(Code of Chinese Graphic Character Set for Information Interchange – Primary Set)*. 中华人民共和国国家标准 (National Standard of the People's Republic of China, GB) GB/T 2312-1980. China, Beijing (中国北京市): 中国国家标准化管理委员会 (Standardization Administration of the People's Republic of China, SAC), 1980–1. Mai 1981. URL: https://openstd.samr.gov.cn/bzgk/gb/newGbInfo?hcno=5664A728BD9D523DE3B99BC37AC7A2CC (besucht am 2024-07-26) (siehe S. 277–283).

[82] 信息技术中文编码字符集 *(Information Technology – Chinese Coded Character Set)*. 中华人民共和国国家标准 (National Standard of the People's Republic of China, GB) GB 18030-2022. China, Beijing (中国北京市): 中华人民共和国工业和信息化部 (Ministry of Industry and Information Technology, MIIT): 国家市场监督管理总局 (State Administration for Market Regulation) und 中国国家标准化管理委员会 (Standardization Administration of the People's Republic of China, SAC), 19. Juli 2022–1. Aug. 2023. URL: https://openstd.samr.gov.cn/bzgk/gb/newGbInfo?hcno=A1931A578FE14957104988029B0833D3 (besucht am 2024-07-26) (siehe S. 277–283).

[83] 汉字内码扩展规范 *(Chinese Internal Code Specification, National Standard Extended GBK)*. 中华人民共和国国家标准 (National Standard of the People's Republic of China, GB) 1995 229. China, Beijing (中国北京市): 中华人民共和国全国信息技术标准化技术委员会 (National Information Technology Standardization Technical Committee of the People's Republic of China), 国家技术监督局 (National Bureau of Technical Supervision) und 中华人民共和国电子工业部 (Ministry of Electronics Industry of the People's Republic of China), 15. Dez. 1995 (siehe S. 277–283).

# Glossary (in English) I



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 Christ [9]. 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 Common Era (CE) [50,80].

C is a programming language, which is very successful in system programming situations [18,49,72].

case-insensitive refers to a way to compare text and characters, in which uppercase letters like „A“ or „Z“, respectively, are considered to be equal to their lowercase counterparts such as „a“ and „z“, respectively. It is thus the opposite of case-sensitive comparisons, where the letters are considered to be the different.

case-sensitive refers to a way to compare text and characters, in which uppercase letters like „A“ or „Z“ are considered to be different from lowercase letters like „a“ and „z“. It is thus the opposite of case-insensitive comparisons, where the letters are considered to be the same. In Python, Java, and C, most string operations such as sorting, equality comparison, and finding are case-sensitive.

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 Christ [9]. 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) [50,80].

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* [71].

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 DB [78].

denominator The number $b$ of a fraction $\frac{a}{b} \in \mathbb{Q}$ is called the *denominator*.

irrational An *irrational* number $z$ is a real number that cannot be expressed as fraction $\frac{p}{q}$ of two integers $p \in \mathbb{Z}$ and $q \in \mathbb{N_1}$ [3,74]. In other words, it holds that $z \in \mathbb{R}$ and $z \notin \mathbb{Q}$. The set of irrational numbers could be denoted as $\mathbb{R} \setminus \mathbb{Q}$. Irrational numbers have decimal expansions that neither terminate nor become periodic. Examples of irrational numbers are $\sqrt{2}$, $\phi$, $\pi$, and $e$.

# Glossary (in English) II



**Java** is another very successful programming language, with roots in the C family of languages [4,39].

**Linux** is the leading open source operating system, i.e., a free alternative to Microsoft Windows [2,31,51,65,70]. We recommend using it for this course, for software development, and for research. Learn more at https://www.linux.org. Its variant Ubuntu is particularly easy to use and install.

**Microsoft Windows** is a commercial proprietary operating system [5]. 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.

**numerator** The number $a$ of a fraction $\frac{a}{b} \in \mathbb{Q}$ is called the *numerator*.

**PostgreSQL** An open source object-relational database management system (DBMS) [23,46,48,61]. See https://postgresql.org for more information.

**Python** The Python programming language [33,38,40,73], i.e., what you will learn about in our book [73]. 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 [12,29,30,54,59,71,77].

**SQL** The *Structured Query Language* is basically a programming language for querying and manipulating relational databases [10,14,16,17,34,43,55–57,59]. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference [55].

**transcendental** A *transcendental* number is a number that is not the root of any integer polynomial, meaning that it is not an algebraic number of any degree [76]. Every real transcendental number must also be irrational, since a rational number is, by definition, an algebraic number of degree one [76]. Examples of transcendental numbers are $\pi$ and $e$.

# Glossary (in English) III



**Ubuntu** is a variant of the open source operating system Linux [11,32]. 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.

**Unicode** A standard for assigning characters to numbers [35,64,67]. 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.

$\phi$ The golden ratio (or golden section) $\phi$ is the irrational number $\frac{1+\sqrt{5}}{2}$. It is the ratio of a line segment cut into two pieces of different lengths such that the ratio of the whole segment to that of the longer segment is equal to the ratio of the longer segment to the shorter segment [8,20]. The golden ratio is approximately $\phi \approx 1.618\,033\,988\,749\,894\,848\,204\,586\,834$ [52]. Represented as `float` in Python, its value is `1.618033988749895`.

$\pi$ 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 [24,36,44], which is approximately $\pi \approx 3.141\,592\,653\,589\,793\,238\,462\,643$. An early method to approximate $\pi$ was developed by the Chinese mathematician LIU Hui (刘徽) based on inscribing regular $e$-gons into a circle somewhere in the third century CE [45,79] in his commentary to the famous Chinese mathematics book *Jiu Zhang Suanshu* (九章算术) [13,15,37,45,58]. In Python, $\pi$ is provided by the `math` module as variable `pi` with value `3.141592653589793`. In PostgreSQL, it is provided by the SQL function `pi()` with value `3.141592653589793` [41].

$e$ is Euler's number [22], the base of the natural logarithm. $e \in \mathbb{R}$ is an irrational and transcendental number [24,36], which is approximately $e \approx 2.718\,281\,828\,459\,045\,235\,360$. In Python, it is provided by the `math` module as variable `e` with value `2.718281828459045`. In PostgreSQL, you can obtain it via the SQL function `exp(1)` as value `2.718281828459045` [41].

$\mathbb{N_0}$ the set of the natural numbers *including* 0, i.e., 0, 1, 2, 3, and so on. It holds that $\mathbb{N_0} \subset \mathbb{Z}$ and that $\mathbb{N_0} = \mathbb{N_1} \cup \{0\}$.

$\mathbb{N_1}$ the set of the natural numbers *excluding* 0, i.e., 1, 2, 3, 4, and so on. It holds that $\mathbb{N_1} \subset \mathbb{Z}$ and that $\mathbb{N_1} = \mathbb{N_0} \setminus \{0\}$.

# Glossary (in English) IV

$\mathbb{Q}$ the set of the rational numbers, i.e., the set of all numbers that can be the result of $\frac{a}{b}$ with $a, b \in \mathbb{Z}$ and $b \neq 0$. $a$ is called the numerator and $b$ is called the denominator. It holds that $\mathbb{Z} \subset \mathbb{Q}$ and $\mathbb{Q} \subset \mathbb{R}$.

$\mathbb{R}$ the set of the real numbers.

$\mathbb{Z}$ the set of the integers numbers including positive and negative numbers and 0, i.e., . . . , -3, -2, -1, 0, 1, 2, 3, . . . , and so on. It holds that $\mathbb{Z} \subset \mathbb{R}$.