



# Programming with Python

## 20. Mengen

Thomas Weise (汤卫思)  
[tweise@hfuu.edu.cn](mailto:tweise@hfuu.edu.cn)

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

应用优化研究所  
人工智能与大数据学院  
合肥大学  
中国安徽省合肥市

# Programming with Python



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

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



# Outline

1. Einleitung
2. Beispiele
3. Zusammenfassung





# Einleitung



# Einleitung

- Listen und Tuples sind klassische Sequenz-basierte Datenstrukturen.



# Einleitung

- Listen und Tuples sind klassische Sequenz-basierte Datenstrukturen.
- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).



# Einleitung

- Listen und Tuples sind klassische Sequenz-basierte Datenstrukturen.
- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).
- Listen und Tupels können ein bestimmtes Element beliebig oft beinhalten.



# Einleitung

- Listen und Tuples sind klassische Sequenz-basierte Datenstrukturen.
- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).
- Listen und Tupels können ein bestimmtes Element beliebig oft beinhalten.
- Mengen (EN: *sets*) implementieren dagegen das mathematische Konzept von Mengen.



# Einleitung

- Listen und Tuples sind klassische Sequenz-basierte Datenstrukturen.
- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).
- Listen und Tupels können ein bestimmtes Element beliebig oft beinhalten.
- Mengen (EN: *sets*) implementieren dagegen das mathematische Konzept von Mengen.
- Sie können ein Element höchstens einmal beihalten.



# Einleitung

- Listen und Tuples sind klassische Sequenz-basierte Datenstrukturen.
- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).
- Listen und Tupels können ein bestimmtes Element beliebig oft beinhalten.
- Mengen (EN: *sets*) implementieren dagegen das mathematische Konzept von Mengen.
- Sie können ein Element höchstens einmal beihalten.

## Gute Praxis

Speichern Sie nur unveränderliche Objekte in Mengen.



# Einleitung

- Listen und Tuples sind klassische Sequenz-basierte Datenstrukturen.
- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).
- Listen und Tupels können ein bestimmtes Element beliebig oft beinhalten.
- Mengen (EN: *sets*) implementieren dagegen das mathematische Konzept von Mengen.
- Sie können ein Element höchstens einmal beihalten.
- Man sollte auch nur unveränderliche Objekte in Mengen speichern, denn wenn man veränderliche Objekte in Mengen speichern würde, dann könnte man nicht mehr garantieren, dass ein Element höchstens einmal vorkommt.



# Einleitung

- Listen und Tuples sind klassische Sequenz-basierte Datenstrukturen.
- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).
- Listen und Tupels können ein bestimmtes Element beliebig oft beinhalten.
- Mengen (EN: *sets*) implementieren dagegen das mathematische Konzept von Mengen.
- Sie können ein Element höchstens einmal beihalten.
- Man sollte auch nur unveränderliche Objekte in Mengen speichern, denn wenn man veränderliche Objekte in Mengen speichern würde, dann könnte man nicht mehr garantieren, dass ein Element höchstens einmal vorkommt.
- Mengen selbst sind aber veränderlich.



# Einleitung

- Listen und Tuples sind klassische Sequenz-basierte Datenstrukturen.
- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).
- Listen und Tupels können ein bestimmtes Element beliebig oft beinhalten.
- Mengen (EN: *sets*) implementieren dagegen das mathematische Konzept von Mengen.
- Sie können ein Element höchstens einmal beihalten.
- Man sollte auch nur unveränderliche Objekte in Mengen speichern, denn wenn man veränderliche Objekte in Mengen speichern würde, dann könnte man nicht mehr garantieren, dass ein Element höchstens einmal vorkommt.
- Mengen selbst sind aber veränderlich.

## Gute Praxis

Mengen sind ungeordnet. Machen Sie niemals irgendwelche Annahmen darüber, in welche Reihenfolge Objekte in Mengen gespeichert werden.



# Einleitung

- Listen und Tuples sind klassische Sequenz-basierte Datenstrukturen.
- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).
- Listen und Tupels können ein bestimmtes Element beliebig oft beinhalten.
- Mengen (EN: *sets*) implementieren dagegen das mathematische Konzept von Mengen.
- Sie können ein Element höchstens einmal beihalten.
- Man sollte auch nur unveränderliche Objekte in Mengen speichern, denn wenn man veränderliche Objekte in Mengen speichern würde, dann könnte man nicht mehr garantieren, dass ein Element höchstens einmal vorkommt.
- Mengen selbst sind aber veränderlich.
- Mengen sind ungeordnete Kollektionen<sup>62</sup>.



# Einleitung



- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).
- Listen und Tupels können ein bestimmtes Element beliebig oft beinhalten.
- Mengen (EN: *sets*) implementieren dagegen das mathematische Konzept von Mengen.
- Sie können ein Element höchstens einmal beihalten.
- Man sollte auch nur unveränderliche Objekte in Mengen speichern, denn wenn man veränderliche Objekte in Mengen speichern würde, dann könnte man nicht mehr garantieren, dass ein Element höchstens einmal vorkommt.
- Mengen selbst sind aber veränderlich.
- Mengen sind ungeordnete Kollektionen<sup>62</sup>.
- Deshalb können wir Mengen auch nicht indizieren (anders als Listen oder Tupels).

# Einleitung



- Eine weitere klassische Datenstruktur ist die Menge (EN: *set*).
- Listen und Tupels können ein bestimmtes Element beliebig oft beinhalten.
- Mengen (EN: *sets*) implementieren dagegen das mathematische Konzept von Mengen.
- Sie können ein Element höchstens einmal beihalten.
- Man sollte auch nur unveränderliche Objekte in Mengen speichern, denn wenn man veränderliche Objekte in Mengen speichern würde, dann könnte man nicht mehr garantieren, dass ein Element höchstens einmal vorkommt.
- Mengen selbst sind aber veränderlich.
- Mengen sind ungeordnete Kollektionen<sup>62</sup>.
- Deshalb können wir Mengen auch nicht indizieren (anders als Listen oder Tupels).
- Wahrscheinlich fragen Sie sich nun: *Wofür brauche ich eigentlich Mengen?*



# Wofür wir Mengen brauchen.

- *Wofür brauche ich eigentlich Mengen?*



# Wofür wir Mengen brauchen.

- *Wofür brauche ich eigentlich Mengen?*
- Elemente in eine Kollektion einfügen oder löschen und gucken, ob ein Element in einer Kollektion ist ... das können wir auch mit Listen machen.





## Wofür wir Mengen brauchen.

- *Wofür brauche ich eigentlich Mengen?*
- Elemente in eine Kollektion einfügen oder löschen und gucken, ob ein Element in einer Kollektion ist ... das können wir auch mit Listen machen.
- Und bei Listen können wir über Indizes auf einzelne Elemente zugreifen, bei Mengen nicht.

# Wofür wir Mengen brauchen.



- *Wofür brauche ich eigentlich Mengen?*
- Mengen sind schnell.



# Wofür wir Mengen brauchen.

- *Wofür brauche ich eigentlich Mengen?*
- Mengen sind schnell.
- Mengen sind in Python auf dem Konzept von Hash-Tabellen implementiert und haben daher die Performanz dieser Datenstrukturen<sup>17,39,64</sup>.



# Wofür wir Mengen brauchen.

- *Wofür brauche ich eigentlich Mengen?*
- Mengen sind schnell.
- Mengen sind in Python auf dem Konzept von Hash-Tabellen implementiert und haben daher die Performanz dieser Datenstrukturen<sup>17,39,64</sup>.
- Wenn wir prüfen wollen, ob ein Element  $x$  in einer Menge ist, dann können wir das im Durchschnitt in  $\mathcal{O}(1)$  machen, also in etwa in konstanter Zeit, gleichgültig wie viele Elemente in der Menge sind<sup>1,33,52</sup>.



# Wofür wir Mengen brauchen.

- *Wofür brauche ich eigentlich Mengen?*
- Mengen sind schnell.
- Mengen sind in Python auf dem Konzept von Hash-Tabellen implementiert und haben daher die Performanz dieser Datenstrukturen<sup>17,39,64</sup>.
- Wenn wir prüfen wollen, ob ein Element  $x$  in einer Menge ist, dann können wir das im Durchschnitt in  $\mathcal{O}(1)$  machen, also in etwa in konstanter Zeit, gleichgültig wie viele Elemente in der Menge sind<sup>1,33,52</sup>.
- Bei einer Liste mit  $n$  Elementen dauert das  $\mathcal{O}(n)$  im Durchschnitt, also eine Zeit linear in der Länge der Liste.

# Wofür wir Mengen brauchen.



- *Wofür brauche ich eigentlich Mengen?*
- Mengen sind schnell.
- Mengen sind in Python auf dem Konzept von Hash-Tabellen implementiert und haben daher die Performanz dieser Datenstrukturen<sup>17,39,64</sup>.
- Wenn wir prüfen wollen, ob ein Element  $x$  in einer Menge ist, dann können wir das im Durchschnitt in  $\mathcal{O}(1)$  machen, also in etwa in konstanter Zeit, gleichgültig wie viele Elemente in der Menge sind<sup>1,33,52</sup>.
- Bei einer Liste mit  $n$  Elementen dauert das  $\mathcal{O}(n)$  im Durchschnitt, also eine Zeit linear in der Länge der Liste.
- Wenn wir nur wenige Elemente haben oder eine Reihenfolge der Elemente brauchen, dann nehmen wir Listen.

# Wofür wir Mengen brauchen.



- *Wofür brauche ich eigentlich Mengen?*
- Mengen sind schnell.
- Mengen sind in Python auf dem Konzept von Hash-Tabellen implementiert und haben daher die Performanz dieser Datenstrukturen<sup>17,39,64</sup>.
- Wenn wir prüfen wollen, ob ein Element  $x$  in einer Menge ist, dann können wir das im Durchschnitt in  $\mathcal{O}(1)$  machen, also in etwa in konstanter Zeit, gleichgültig wie viele Elemente in der Menge sind<sup>1,33,52</sup>.
- Bei einer Liste mit  $n$  Elementen dauert das  $\mathcal{O}(n)$  im Durchschnitt, also eine Zeit linear in der Länge der Liste.
- Wenn wir nur wenige Elemente haben oder eine Reihenfolge der Elemente brauchen, dann nehmen wir Listen.
- Wenn wir mehr Elemente haben und effizient prüfen wollen, ob ein Element beinhaltet ist, oder wenn wir effizient Mengenoperationen ausführen wollen, dann nehmen wir Mengen.



## Beispiele



# Erstellen, Type Hints, Einfache Operationen

- Mengen-Literale werden mit geschweiften Klammern definiert.

```
1  """An example of creating, modifying, and converting sets."""
2
3  upper: set[str] = {"A", "G", "B", "T", "V"} # Some uppercase letters...
4  print(f"some uppercase letters are: {upper}") # Print the set.
5
6  upper.add("Z") # Add the letter "Z" to the set.
7  upper.add("A") # The letter "A" is already in the set.
8  upper.add("Z") # The letter "Z" is already in the set.
9  print(f"some more uppercase letters are: {upper}") # Print the set.
10
11 upper.update(["K", "G", "W", "Q", "W"]) # Try to add 5 letters.
12 print(f"even more uppercase letters are: {upper}") # Print the set.
13
14 lower_tuple: tuple[str, ...] = ("b", "i", "j", "c", "t", "i")
15 lower: set[str] = set(lower_tuple) # Convert a tuple to a set.
16 print(f"some lowercase letters are: {lower}") # Print the set 'lower'.
17 lower.remove("b") # Delete letter b from the set of lower case letters.
18 print(f"lowercase letters after deleting 'b': {lower}") # Print the set
19
20 letters: set[str] = set(lower) # Copy the set of lowercase characters.
21 letters.update(upper) # Add all uppercase characters.
22 print(f"some letters are: {letters}") # Print the set 'letters'.
23
24 # Create a sorted list containing all elements of the set letters.
25 # Warning: Strings are sorted such that uppercase characters come before
26 # lowercase characters.
27 letters_list: list[str] = sorted(letters)
28 print(f"the sorted list of letters is: {letters_list}")
```

# Erstellen, Type Hints, Einfache Operationen

- Mengen-Literale werden mit geschweiften Klammern definiert.
- Als Type-Hint wird `set[T]` verwendet, wobei `T` der Datentyp für die Elemente der Menge ist (und `set` der Datentyp für Mengen).

```
1  """An example of creating, modifying, and converting sets."""
2
3  upper: set[str] = {"A", "G", "B", "T", "V"} # Some uppercase letters...
4  print(f"some uppercase letters are: {upper}") # Print the set.
5
6  upper.add("Z") # Add the letter "Z" to the set.
7  upper.add("A") # The letter "A" is already in the set.
8  upper.add("Z") # The letter "Z" is already in the set.
9  print(f"some more uppercase letters are: {upper}") # Print the set.
10
11 upper.update(["K", "G", "W", "Q", "W"]) # Try to add 5 letters.
12 print(f"even more uppercase letters are: {upper}") # Print the set.
13
14 lower_tuple: tuple[str, ...] = ("b", "i", "j", "c", "t", "i")
15 lower: set[str] = set(lower_tuple) # Convert a tuple to a set.
16 print(f"some lowercase letters are: {lower}") # Print the set 'lower'.
17 lower.remove("b") # Delete letter b from the set of lower case letters.
18 print(f"lowercase letters after deleting 'b': {lower}") # Print the set
19
20 letters: set[str] = set(lower) # Copy the set of lowercase characters.
21 letters.update(upper) # Add all uppercase characters.
22 print(f"some letters are: {letters}") # Print the set 'letters'.
23
24 # Create a sorted list containing all elements of the set letters.
25 # Warning: Strings are sorted such that uppercase characters come before
26 # lowercase characters.
27 letters_list: list[str] = sorted(letters)
28 print(f"the sorted list of letters is: {letters_list}")
```

# Erstellen, Type Hints, Einfache Operationen

- Mengen-Literale werden mit geschweiften Klammern definiert.
- Als Type-Hint wird `set[T]` verwendet, wobei `T` der Datentyp für die Elemente der Menge ist (und `set` der Datentyp für Mengen).
- Mit `s.add(e)` können wir das Element `e` in die Menge `s` einfügen.

```
1  """An example of creating, modifying, and converting sets."""
2
3  upper: set[str] = {"A", "G", "B", "T", "V"} # Some uppercase letters...
4  print(f"some uppercase letters are: {upper}") # Print the set.
5
6  upper.add("Z") # Add the letter "Z" to the set.
7  upper.add("A") # The letter "A" is already in the set.
8  upper.add("Z") # The letter "Z" is already in the set.
9  print(f"some more uppercase letters are: {upper}") # Print the set.
10
11 upper.update(["K", "G", "W", "Q", "W"]) # Try to add 5 letters.
12 print(f"even more uppercase letters are: {upper}") # Print the set.
13
14 lower_tuple: tuple[str, ...] = ("b", "i", "j", "c", "t", "i")
15 lower: set[str] = set(lower_tuple) # Convert a tuple to a set.
16 print(f"some lowercase letters are: {lower}") # Print the set 'lower'.
17 lower.remove("b") # Delete letter b from the set of lower case letters.
18 print(f"lowercase letters after deleting 'b': {lower}") # Print the set
19
20 letters: set[str] = set(lower) # Copy the set of lowercase characters.
21 letters.update(upper) # Add all uppercase characters.
22 print(f"some letters are: {letters}") # Print the set 'letters'.
23
24 # Create a sorted list containing all elements of the set letters.
25 # Warning: Strings are sorted such that uppercase characters come before
26 # lowercase characters.
27 letters_list: list[str] = sorted(letters)
28 print(f"the sorted list of letters is: {letters_list}")
```

# Erstellen, Type Hints, Einfache Operationen

- Mengen-Literale werden mit geschweiften Klammern definiert.
- Als Type-Hint wird `set[T]` verwendet, wobei `T` der Datentyp für die Elemente der Menge ist (und `set` der Datentyp für Mengen).
- Mit `s.add(e)` können wir das Element `e` in die Menge `s` einfügen.
- Mit `s.update(c)` fügen wir alle Elemente der Kollektion `c` in die Menge `s` ein.

```
1  """An example of creating, modifying, and converting sets."""
2
3  upper: set[str] = {"A", "G", "B", "T", "V"} # Some uppercase letters...
4  print(f"some uppercase letters are: {upper}") # Print the set.
5
6  upper.add("Z") # Add the letter "Z" to the set.
7  upper.add("A") # The letter "A" is already in the set.
8  upper.add("Z") # The letter "Z" is already in the set.
9  print(f"some more uppercase letters are: {upper}") # Print the set.
10
11 upper.update(["K", "G", "W", "Q", "W"]) # Try to add 5 letters.
12 print(f"even more uppercase letters are: {upper}") # Print the set.
13
14 lower_tuple: tuple[str, ...] = ("b", "i", "j", "c", "t", "i")
15 lower: set[str] = set(lower_tuple) # Convert a tuple to a set.
16 print(f"some lowercase letters are: {lower}") # Print the set 'lower'.
17 lower.remove("b") # Delete letter b from the set of lower case letters.
18 print(f"lowercase letters after deleting 'b': {lower}") # Print the set
19
20 letters: set[str] = set(lower) # Copy the set of lowercase characters.
21 letters.update(upper) # Add all uppercase characters.
22 print(f"some letters are: {letters}") # Print the set 'letters'.
23
24 # Create a sorted list containing all elements of the set letters.
25 # Warning: Strings are sorted such that uppercase characters come before
26 # lowercase characters.
27 letters_list: list[str] = sorted(letters)
28 print(f"the sorted list of letters is: {letters_list}")
```

# Erstellen, Type Hints, Einfache Operationen

- Mengen-Literale werden mit geschweiften Klammern definiert.
- Als Type-Hint wird `set[T]` verwendet, wobei `T` der Datentyp für die Elemente der Menge ist (und `set` der Datentyp für Mengen).
- Mit `s.add(e)` können wir das Element `e` in die Menge `s` einfügen.
- Mit `s.update(c)` fügen wir alle Elemente der Kollektion `c` in die Menge `s` ein.
- Mit `set(c)` erstellen wir eine Menge mit den Elementen der Kollektion `c`.

```
1 """An example of creating, modifying, and converting sets."""
2
3 upper: set[str] = {"A", "G", "B", "T", "V"} # Some uppercase letters...
4 print(f"some uppercase letters are: {upper}") # Print the set.
5
6 upper.add("Z") # Add the letter "Z" to the set.
7 upper.add("A") # The letter "A" is already in the set.
8 upper.add("Z") # The letter "Z" is already in the set.
9 print(f"some more uppercase letters are: {upper}") # Print the set.
10
11 upper.update(["K", "G", "W", "Q", "W"]) # Try to add 5 letters.
12 print(f"even more uppercase letters are: {upper}") # Print the set.
13
14 lower_tuple: tuple[str, ...] = ("b", "i", "j", "c", "t", "i")
15 lower: set[str] = set(lower_tuple) # Convert a tuple to a set.
16 print(f"some lowercase letters are: {lower}") # Print the set 'lower'.
17 lower.remove("b") # Delete letter b from the set of lower case letters.
18 print(f"lowercase letters after deleting 'b': {lower}") # Print the set
19
20 letters: set[str] = set(lower) # Copy the set of lowercase characters.
21 letters.update(upper) # Add all uppercase characters.
22 print(f"some letters are: {letters}") # Print the set 'letters'.
23
24 # Create a sorted list containing all elements of the set letters.
25 # Warning: Strings are sorted such that uppercase characters come before
26 # lowercase characters.
27 letters_list: list[str] = sorted(letters)
28 print(f"the sorted list of letters is: {letters_list}")
```

# Erstellen, Type Hints, Einfache Operationen

- Als Type-Hint wird `set[T]` verwendet, wobei `T` der Datentyp für die Elemente der Menge ist (und `set` der Datentyp für Mengen).
- Mit `s.add(e)` können wir das Element `e` in die Menge `s` einfügen.
- Mit `s.update(c)` fügen wir alle Elemente der Kollektion `c` in die Menge `s` ein.
- Mit `set(c)` erstellen wir eine Menge mit den Elementen der Kollektion `c`.
- `s.remove(e)` entfernt das Element `e` aus der Menge `s`.

```
1  """An example of creating, modifying, and converting sets."""
2
3  upper: set[str] = {"A", "G", "B", "T", "V"} # Some uppercase letters...
4  print(f"some uppercase letters are: {upper}") # Print the set.
5
6  upper.add("Z") # Add the letter "Z" to the set.
7  upper.add("A") # The letter "A" is already in the set.
8  upper.add("Z") # The letter "Z" is already in the set.
9  print(f"some more uppercase letters are: {upper}") # Print the set.
10
11 upper.update(["K", "G", "W", "Q", "W"]) # Try to add 5 letters.
12 print(f"even more uppercase letters are: {upper}") # Print the set.
13
14 lower_tuple: tuple[str, ...] = ("b", "i", "j", "c", "t", "i")
15 lower: set[str] = set(lower_tuple) # Convert a tuple to a set.
16 print(f"some lowercase letters are: {lower}") # Print the set 'lower'.
17 lower.remove("b") # Delete letter b from the set of lower case letters.
18 print(f"lowercase letters after deleting 'b': {lower}") # Print the set
19
20 letters: set[str] = set(lower) # Copy the set of lowercase characters.
21 letters.update(upper) # Add all uppercase characters.
22 print(f"some letters are: {letters}") # Print the set 'letters'.
23
24 # Create a sorted list containing all elements of the set letters.
25 # Warning: Strings are sorted such that uppercase characters come before
26 # lowercase characters.
27 letters_list: list[str] = sorted(letters)
28 print(f"the sorted list of letters is: {letters_list}")
```

# Erstellen, Type Hints, Einfache Operationen

- Mit `s.add(e)` können wir das Element `e` in die Menge `s` einfügen.
- Mit `s.update(c)` fügen wir alle Elemente der Kollektion `c` in die Menge `s` ein.
- Mit `set(c)` erstellen wir eine Menge mit den Elementen der Kollektion `c`.
- `s.remove(e)` entfernt das Element `e` aus der Menge `s`.
- Randnotiz: Die Listen-Funktion `sorted(c)` erstellt eine sortierte Liste aus den Elementen der Kollektion `c`.

```
1  """An example of creating, modifying, and converting sets."""
2
3  upper: set[str] = {"A", "G", "B", "T", "V"} # Some uppercase letters...
4  print(f"some uppercase letters are: {upper}") # Print the set.
5
6  upper.add("Z") # Add the letter "Z" to the set.
7  upper.add("A") # The letter "A" is already in the set.
8  upper.add("Z") # The letter "Z" is already in the set.
9  print(f"some more uppercase letters are: {upper}") # Print the set.
10
11 upper.update(["K", "G", "W", "Q", "W"]) # Try to add 5 letters.
12 print(f"even more uppercase letters are: {upper}") # Print the set.
13
14 lower_tuple: tuple[str, ...] = ("b", "i", "j", "c", "t", "i")
15 lower: set[str] = set(lower_tuple) # Convert a tuple to a set.
16 print(f"some lowercase letters are: {lower}") # Print the set 'lower'.
17 lower.remove("b") # Delete letter b from the set of lower case letters.
18 print(f"lowercase letters after deleting 'b': {lower}") # Print the set
19
20 letters: set[str] = set(lower) # Copy the set of lowercase characters.
21 letters.update(upper) # Add all uppercase characters.
22 print(f"some letters are: {letters}") # Print the set 'letters'.
23
24 # Create a sorted list containing all elements of the set letters.
25 # Warning: Strings are sorted such that uppercase characters come before
26 # lowercase characters.
27 letters_list: list[str] = sorted(letters)
28 print(f"the sorted list of letters is: {letters_list}")
```

# Erstellen, Type Hints, Einfache Operationen



- Mit `set(c)` erstellen wir eine Menge mit den Elementen der Kollektion `c`.
- `s.remove(e)` entfernt das Element `e` aus der Menge `s`.
- Randnotiz: Die Listen-Funktion `sorted(c)` erstellt eine sortierte Liste aus den Elementen der Kollektion `c`.

## Gute Praxis

Vorsicht beim Vergleichen von Strings: Standardmäßig kommen Großbuchstaben vor Kleinbuchstaben, also `"A" < "a"`. Wenn Sie Groß- und Kleinbuchstaben gleich behandeln wollen, dann müssen Sie sowas wie `sorted(my_text, key=str.casefold)` machen.

# Erstellen, Type Hints, Einfache Operationen

- Mit `s.add(e)` können wir das Element `e` in die Menge `s` einfügen.
- Mit `s.update(c)` fügen wir alle Elemente der Kollektion `c` in die Menge `s` ein.
- Mit `set(c)` erstellen wir eine Menge mit den Elementen der Kollektion `c`.
- `s.remove(e)` entfernt das Element `e` aus der Menge `s`.
- Randnotiz: Die Listen-Funktion `sorted(c)` erstellt eine sortierte Liste aus den Elementen der Kollektion `c`.

```
1 """An example of creating, modifying, and converting sets."""
2
3 upper: set[str] = {"A", "G", "B", "T", "V"} # Some uppercase letters...
4 print(f"some uppercase letters are: {upper}") # Print the set.
5
6 upper.add("Z") # Add the letter "Z" to the set.
7 upper.add("A") # The letter "A" is already in the set.
8 upper.add("Z") # The letter "Z" is already in the set.
9 print(f"some more uppercase letters are: {upper}") # Print the set.
10
11 upper.update(["K", "G", "W", "Q", "W"]) # Try to add 5 letters.
12 print(f"even more uppercase letters are: {upper}") # Print the set.
13
14 lower_tuple: tuple[str, ...] = ("b", "i", "j", "c", "t", "i")
15 lower: set[str] = set(lower_tuple) # Convert a tuple to a set.
16 print(f"some lowercase letters are: {lower}") # Print the set 'lower'.
17 lower.remove("b") # Delete letter b from the set of lower case letters.
18 print(f"lowercase letters after deleting 'b': {lower}") # Print the set
19
20 letters: set[str] = set(lower) # Copy the set of lowercase characters.
21 letters.update(upper) # Add all uppercase characters.
22 print(f"some letters are: {letters}") # Print the set 'letters'.
23
24 # Create a sorted list containing all elements of the set letters.
25 # Warning: Strings are sorted such that uppercase characters come before
26 # lowercase characters.
27 letters_list: list[str] = sorted(letters)
28 print(f"the sorted list of letters is: {letters_list}")

↓ python3 sets_1.py ↓
```

```
1 some uppercase letters are: {'T', 'G', 'A', 'B', 'V'}
2 some more uppercase letters are: {'T', 'G', 'A', 'B', 'V', 'Z'}
3 even more uppercase letters are: {'W', 'G', 'T', 'A', 'B', 'Z', 'K', 'V'
4   ↪ ', 'Q'}
5 some lowercase letters are: {'b', 'i', 't', 'j', 'c'}
6 lowercase letters after deleting 'b': {'i', 't', 'j', 'c'}
7 some letters are: {'W', 'G', 'Z', 'Q', 'V', 'T', 'i', 'A', 't', 'B', 'j'
8   ↪ ', 'c', 'K'}
9 the sorted list of letters is: ['A', 'B', 'G', 'K', 'Q', 'T', 'V', 'W',
  ↪ 'Z', 'c', 'i', 'j', 't']
```

# Erstellen, Type Hints, Einfache Objekte

- Mit `s.add(e)` können wir das Element `e` in die Menge `s` einfügen.
- Mit `s.update(c)` fügen wir alle Elemente der Kollektion `c` in die Menge `s` ein.
- Mit `set(c)` erstellen wir eine Menge mit den Elementen der Kollektion `c`.
- `s.remove(e)` entfernt das Element `e` aus der Menge `s`.
- Randnotiz: Die Listerfunktion `sorted(c)` erstellt eine Liste aus den Elementen von `c`.

```
1     """An example of creating, modifying, and converting sets."""
2
3     upper: set[str] = {"A", "G", "B", "T", "V"} # Some uppercase letters...
4     print(f"some uppercase letters are: {upper}") # Print the set.
5
6     upper.add("Z") # Add the letter "Z" to the set.
7     upper.add("A") # The letter "A" is already in the set.
8     upper.add("Z") # The letter "Z" is already in the set.
9     print(f"some more uppercase letters are: {upper}") # Print the set.
10
11    upper.update(["K", "G", "W", "Q", "W"]) # Try to add 5 letters.
12    print(f"even more uppercase letters are: {upper}") # Print the set.
13
14    lower_tuple: tuple[str, ...] = ("b", "i", "j", "c", "t", "i")
15    lower: set[str] = set(lower_tuple) # Convert a tuple to a set.
16    print(f"some lowercase letters are: {lower}") # Print the set 'lower'.
17    lower.remove("b") # Delete letter b from the set of lower case letters.
18    print(f"lowercase letters after deleting 'b': {lower}") # Print the set.
19
20    letters: set[str] = set(lower) # Copy the set of lowercase characters.
21    letters.update(upper) # Add all uppercase characters.
22    print(f"some letters are: {letters}") # Print the set 'letters'.
```

```
1 some uppercase letters are: {'T', 'G', 'A', 'B', 'V'}
2 some more uppercase letters are: {'T', 'G', 'A', 'B', 'V', 'Z'}
3 even more uppercase letters are: {'W', 'G', 'T', 'A', 'B', 'Z', 'K', 'V'
4     ↪ ', 'Q'}
5 some lowercase letters are: {'b', 'i', 't', 'j', 'c'}
6 lowercase letters after deleting 'b': {'i', 't', 'j', 'c'}
7 some letters are: {'W', 'G', 'Z', 'Q', 'V', 'T', 'i', 'A', 't', 'B', 'j
8     ↪ ', 'c', 'K'}
9 the sorted list of letters is: ['A', 'B', 'G', 'K', 'Q', 'T', 'V', 'W',
10    ↪ 'Z', 'c', 'i', 'j', 't']
```

before

# Mengen Operationen

- Schauen wir uns grundlegende Mengenoperationen an.



# Mengen Operationen



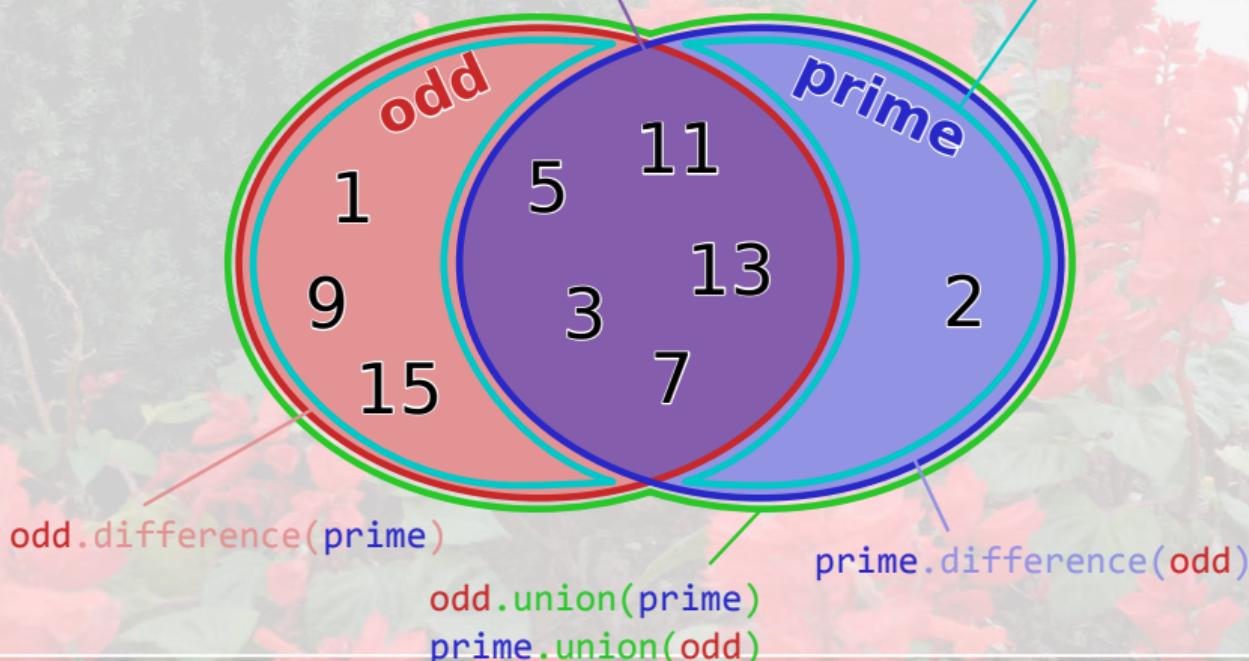
- Schauen wir uns grundlegende Mengenoperationen an.

`odd.intersection(prime)`

`prime.intersection(odd)`

`odd.symmetric_difference(prime)`

`prime.symmetric_difference(odd)`



# Mengen Operationen

- `in` und `not in` funktionieren genau wie bei Listen und Tupeln (nur schneller).

```
1     """An example of creating sets and set operations."""
2
3 odd: set[int] = {1, 3, 5, 7, 9, 11, 13, 15} # a subset of odd numbers
4 print(f"some odd numbers are: {odd}") # Print the set.
5 print(f"is 3 \u2208 odd: {3 in odd}") # Check if 3 is in the set odd.
6 print(f"is 2 \u2209 odd: {2 not in odd}") # Check if 2 is NOT in odd.
7
8 prime: set[int] = {2, 3, 5, 7, 11, 13} # a subset of the prime numbers
9 print(f"some prime numbers are: {prime}") # Print the set.
10
11 set_or: set[int] = odd.union(prime) # Create a new set as union of both
12 print(f"{len(set_or)} numbers are in odd \u222A prime: {set_or},")
13
14 set_and: set[int] = odd.intersection(prime) # Compute the intersection.
15 print(f"{len(set_and)} are in odd \u2229 prime: {set_and},")
16
17 only_prime: set[int] = prime.difference(odd) # Prime but not odd
18 print(f"{len(only_prime)} are in prime \u2216 odd: {only_prime},")
19
20 # Get the numbers that are in one and only one of the two sets.
21 set_xor: set[int] = odd.symmetric_difference(prime)
22 print(f"{len(set_xor)} are in (odd \u222A prime) "
23      f"\u2216 (odd \u2229 prime): {set_xor}, and")
24
25 only_odd: set[int] = odd.difference(prime) # Odd but not prime
26 print(f"{len(only_odd)} are in odd \u2216 prime: {only_odd},")
27 odd.difference_update(prime) # delete all prime numbers from odd
28 print(f"after deleting all primes from odd, we get {odd}")
```

# Mengen Operationen

- `in` und `not in` funktionieren genau wie bei Listen und Tupeln (nur schneller).
- `s1.union(s2)` erstellt eine neue Menge mit allen Elementen aus `s1` und `s2`.

```
1     """An example of creating sets and set operations."""
2
3 odd: set[int] = {1, 3, 5, 7, 9, 11, 13, 15} # a subset of odd numbers
4 print(f"some odd numbers are: {odd}") # Print the set.
5 print(f"is 3 \u2208 odd: {3 in odd}") # Check if 3 is in the set odd.
6 print(f"is 2 \u2209 odd: {2 not in odd}") # Check if 2 is NOT in odd.
7
8 prime: set[int] = {2, 3, 5, 7, 11, 13} # a subset of the prime numbers
9 print(f"some prime numbers are: {prime}") # Print the set.
10
11 set_or: set[int] = odd.union(prime) # Create a new set as union of both
12 print(f"{len(set_or)} numbers are in odd \u222A prime: {set_or},")
13
14 set_and: set[int] = odd.intersection(prime) # Compute the intersection.
15 print(f"{len(set_and)} are in odd \u2229 prime: {set_and},")
16
17 only_prime: set[int] = prime.difference(odd) # Prime but not odd
18 print(f"{len(only_prime)} are in prime \u2216 odd: {only_prime},")
19
20 # Get the numbers that are in one and only one of the two sets.
21 set_xor: set[int] = odd.symmetric_difference(prime)
22 print(f"{len(set_xor)} are in (odd \u222A prime) "
23      f"\u2216 (odd \u2229 prime): {set_xor}, and")
24
25 only_odd: set[int] = odd.difference(prime) # Odd but not prime
26 print(f"{len(only_odd)} are in odd \u2216 prime: {only_odd},")
27 odd.difference_update(prime) # delete all prime numbers from odd
28 print(f"after deleting all primes from odd, we get {odd}")
```

# Mengen Operationen

- `in` und `not in` funktionieren genau wie bei Listen und Tupeln (nur schneller).
- `s1.union(s2)` erstellt eine neue Menge mit allen Elementen aus `s1` und `s2`.
- `s1.intersection(s2)` erstellt eine neue Menge mit allen Elementen die sowohl in `s1` als auch in `s2` sind.

```
1     """An example of creating sets and set operations."""
2
3 odd: set[int] = {1, 3, 5, 7, 9, 11, 13, 15} # a subset of odd numbers
4 print(f"some odd numbers are: {odd}") # Print the set.
5 print(f"is 3 \u2208 odd: {3 in odd}") # Check if 3 is in the set odd.
6 print(f"is 2 \u2209 odd: {2 not in odd}") # Check if 2 is NOT in odd.
7
8 prime: set[int] = {2, 3, 5, 7, 11, 13} # a subset of the prime numbers
9 print(f"some prime numbers are: {prime}") # Print the set.
10
11 set_or: set[int] = odd.union(prime) # Create a new set as union of both
12 print(f"{len(set_or)} numbers are in odd \u222A prime: {set_or},")
13
14 set_and: set[int] = odd.intersection(prime) # Compute the intersection.
15 print(f"{len(set_and)} are in odd \u2229 prime: {set_and},")
16
17 only_prime: set[int] = prime.difference(odd) # Prime but not odd
18 print(f"{len(only_prime)} are in prime \u2216 odd: {only_prime},")
19
20 # Get the numbers that are in one and only one of the two sets.
21 set_xor: set[int] = odd.symmetric_difference(prime)
22 print(f"{len(set_xor)} are in (odd \u222A prime) "
23      f"\u2216 (odd \u2229 prime): {set_xor}, and")
24
25 only_odd: set[int] = odd.difference(prime) # Odd but not prime
26 print(f"{len(only_odd)} are in odd \u2216 prime: {only_odd},")
27 odd.difference_update(prime) # delete all prime numbers from odd
28 print(f"after deleting all primes from odd, we get {odd}")
```

# Mengen Operationen

- `in` und `not in` funktionieren genau wie bei Listen und Tupeln (nur schneller).
- `s1.union(s2)` erstellt eine neue Menge mit allen Elementen aus `s1` und `s2`.
- `s1.intersection(s2)` erstellt eine neue Menge mit allen Elementen die *sowohl* in `s1` *als auch* in `s2` sind.
- `s1.difference(s2)` erstellt eine neue Menge mit allen Elementen die in `s1` *aber nicht* in `s2` sind.

```
1     """An example of creating sets and set operations."""
2
3 odd: set[int] = {1, 3, 5, 7, 9, 11, 13, 15} # a subset of odd numbers
4 print(f"some odd numbers are: {odd}") # Print the set.
5 print(f"is 3 \u2208 odd: {3 in odd}") # Check if 3 is in the set odd.
6 print(f"is 2 \u2209 odd: {2 not in odd}") # Check if 2 is NOT in odd.
7
8 prime: set[int] = {2, 3, 5, 7, 11, 13} # a subset of the prime numbers
9 print(f"some prime numbers are: {prime}") # Print the set.
10
11 set_or: set[int] = odd.union(prime) # Create a new set as union of both
12 print(f"{len(set_or)} numbers are in odd \u222A prime: {set_or},")
13
14 set_and: set[int] = odd.intersection(prime) # Compute the intersection.
15 print(f"{len(set_and)} are in odd \u2229 prime: {set_and},")
16
17 only_prime: set[int] = prime.difference(odd) # Prime but not odd
18 print(f"{len(only_prime)} are in prime \u2216 odd: {only_prime},")
19
20 # Get the numbers that are in one and only one of the two sets.
21 set_xor: set[int] = odd.symmetric_difference(prime)
22 print(f"{len(set_xor)} are in (odd \u222A prime) "
23      f"\u2216 (odd \u2229 prime): {set_xor}, and")
24
25 only_odd: set[int] = odd.difference(prime) # Odd but not prime
26 print(f"{len(only_odd)} are in odd \u2216 prime: {only_odd},")
27 odd.difference_update(prime) # delete all prime numbers from odd
28 print(f"after deleting all primes from odd, we get {odd}")
```

# Mengen Operationen

- `in` und `not in` funktionieren genau wie bei Listen und Tupeln (nur schneller).
- `s1.union(s2)` erstellt eine neue Menge mit allen Elementen aus `s1` und `s2`.
- `s1.intersection(s2)` erstellt eine neue Menge mit allen Elementen die *sowohl* in `s1` als auch in `s2` sind.
- `s1.difference(s2)` erstellt eine neue Menge mit allen Elementen die in `s1` aber nicht in `s2` sind.
- `s1.symmetric_difference(s2)` erstellt eine neue Menge mit allen Elementen die *entweder nur* in `s1` oder *nur* in `s2` sind.

```
1     """An example of creating sets and set operations."""
2
3 odd: set[int] = {1, 3, 5, 7, 9, 11, 13, 15} # a subset of odd numbers
4 print(f"some odd numbers are: {odd}") # Print the set.
5 print(f"is 3 \u2208 odd: {3 in odd}") # Check if 3 is in the set odd.
6 print(f"is 2 \u2209 odd: {2 not in odd}") # Check if 2 is NOT in odd.
7
8 prime: set[int] = {2, 3, 5, 7, 11, 13} # a subset of the prime numbers
9 print(f"some prime numbers are: {prime}") # Print the set.
10
11 set_or: set[int] = odd.union(prime) # Create a new set as union of both
12 print(f"{len(set_or)} numbers are in odd \u222A prime: {set_or},")
13
14 set_and: set[int] = odd.intersection(prime) # Compute the intersection.
15 print(f"{len(set_and)} are in odd \u2229 prime: {set_and},")
16
17 only_prime: set[int] = prime.difference(odd) # Prime but not odd
18 print(f"{len(only_prime)} are in prime \u2216 odd: {only_prime},")
19
20 # Get the numbers that are in one and only one of the two sets.
21 set_xor: set[int] = odd.symmetric_difference(prime)
22 print(f"{len(set_xor)} are in (odd \u222A prime) "
23      f"\u2216 (odd \u2229 prime): {set_xor}, and")
24
25 only_odd: set[int] = odd.difference(prime) # Odd but not prime
26 print(f"{len(only_odd)} are in odd \u2216 prime: {only_odd},")
27 odd.difference_update(prime) # delete all prime numbers from odd
28 print(f"after deleting all primes from odd, we get {odd}")
```

# Mengen Operationen

- `s1.union(s2)` erstellt eine neue Menge mit allen Elementen aus `s1` und `s2`.
- `s1.intersection(s2)` erstellt eine neue Menge mit allen Elementen die *sowohl* in `s1` *als auch* in `s2` sind.
- `s1.difference(s2)` erstellt eine neue Menge mit allen Elementen die in `s1` *aber nicht* in `s2` sind.
- `s1.symmetric_difference(s2)` erstellt eine neue Menge mit allen Elementen die *entweder nur* in `s1` *oder nur* in `s2` sind.
- `len(s)` gibt uns die Anzahl der Elemente in Menge `s`.

```
1     """An example of creating sets and set operations."""
2
3 odd: set[int] = {1, 3, 5, 7, 9, 11, 13, 15} # a subset of odd numbers
4 print(f"some odd numbers are: {odd}") # Print the set.
5 print(f"is 3 \u2208 odd: {3 in odd}") # Check if 3 is in the set odd.
6 print(f"is 2 \u2209 odd: {2 not in odd}") # Check if 2 is NOT in odd.
7
8 prime: set[int] = {2, 3, 5, 7, 11, 13} # a subset of the prime numbers
9 print(f"some prime numbers are: {prime}") # Print the set.
10
11 set_or: set[int] = odd.union(prime) # Create a new set as union of both
12 print(f"{len(set_or)} numbers are in odd \u222A prime: {set_or},")
13
14 set_and: set[int] = odd.intersection(prime) # Compute the intersection.
15 print(f"{len(set_and)} are in odd \u2229 prime: {set_and},")
16
17 only_prime: set[int] = prime.difference(odd) # Prime but not odd
18 print(f"{len(only_prime)} are in prime \u2216 odd: {only_prime},")
19
20 # Get the numbers that are in one and only one of the two sets.
21 set_xor: set[int] = odd.symmetric_difference(prime)
22 print(f"{len(set_xor)} are in (odd \u222A prime) "
23      f"\u2216 (odd \u2229 prime): {set_xor}, and")
24
25 only_odd: set[int] = odd.difference(prime) # Odd but not prime
26 print(f"{len(only_odd)} are in odd \u2216 prime: {only_odd},")
27 odd.difference_update(prime) # delete all prime numbers from odd
28 print(f"after deleting all primes from odd, we get {odd}")
```

# Mengen Operationen

- `s1.union(s2)` erstellt eine neue Menge mit allen Elementen aus `s1` und `s2`.
- `s1.intersection(s2)` erstellt eine neue Menge mit allen Elementen die *sowohl* in `s1` als auch in `s2` sind.
- `s1.difference(s2)` erstellt eine neue Menge mit allen Elementen die in `s1` aber nicht in `s2` sind.
- `s1.symmetric_difference(s2)` erstellt eine neue Menge mit allen Elementen die *entweder nur* in `s1` oder *nur* in `s2` sind.
- `len(s)` gibt uns die Anzahl der Elemente in Menge `s`.

```
1     """An example of creating sets and set operations."""
2
3 odd: set[int] = {1, 3, 5, 7, 9, 11, 13, 15} # a subset of odd numbers
4 print(f"some odd numbers are: {odd}") # Print the set.
5 print(f"Is 3 ∈ odd: {3 in odd}") # Check if 3 is in the set odd.
6 print(f"Is 2 ∉ odd: {2 not in odd}") # Check if 2 is NOT in odd.
7
8 prime: set[int] = {2, 3, 5, 7, 11, 13} # a subset of the prime numbers
9 print(f"some prime numbers are: {prime}") # Print the set.
10
11 set_or: set[int] = odd.union(prime) # Create a new set as union of both
12 print(f"\{len(set_or)} numbers are in odd ∪ prime: {set_or},")
13
14 set_and: set[int] = odd.intersection(prime) # Compute the intersection.
15 print(f"\{len(set_and)} are in odd ∩ prime: {set_and},")
16
17 only_prime: set[int] = prime.difference(odd) # Prime but not odd
18 print(f"\{len(only_prime)} are in prime \ odd: {only_prime},")
19
20 # Get the numbers that are in one and only one of the two sets.
21 set_xor: set[int] = odd.symmetric_difference(prime)
22 print(f"\{len(set_xor)} are in (odd ∆ prime) "
23      f"\{len(only_odd)} (odd ∩ prime): {set_xor}, and")
24
25 only_odd: set[int] = odd.difference(prime) # Odd but not prime
26 print(f"\{len(only_odd)} are in odd \ prime: {only_odd},")
27 odd.difference_update(prime) # delete all prime numbers from odd
28 print(f"after deleting all primes from odd, we get {odd}")
```

↓ python3 sets\_2.py ↓

```
1 some odd numbers are: {1, 3, 5, 7, 9, 11, 13, 15}
2 Is 3 ∈ odd: True
3 Is 2 ∉ odd: True
4 some prime numbers are: {2, 3, 5, 7, 11, 13}
5 9 numbers are in odd ∪ prime: {1, 2, 3, 5, 7, 9, 11, 13, 15},
6 5 are in odd ∩ prime: {3, 5, 7, 11, 13},
7 1 are in prime \ odd: {2},
8 4 are in (odd ∪ prime) \ (odd ∩ prime): {1, 2, 9, 15}, and
9 3 are in odd \ prime: {1, 9, 15},
10 after deleting all primes from odd, we get {1, 9, 15}
```

# Mengen Operationen



- `s1.intersection(s2)` erstellt eine neue Menge mit allen Elementen die *sowohl* in `s1` als auch in `s2` sind.
- `s1.difference(s2)` erstellt eine neue Menge mit allen Elementen die in `s1` aber nicht in `s2` sind.
- `s1.symmetric_difference(s2)` erstellt eine neue Menge mit allen Elementen die *entweder nur* in `s1` oder *nur* in `s2` sind.
- `len(s)` gibt uns die Anzahl der Elemente in Menge `s`.
- Sind Ihnen die coolen Unicode-Escapes aufgefallen?



# Zusammenfassung



# Zusammenfassung



- Mengen implementieren viele Operationen sehr effizient.

# Zusammenfassung



- Mengen implementieren viele Operationen sehr effizient.
- Sie sind schnell darin, zu prüfen ob ein Element in der Kollektion ist.

# Zusammenfassung



- Mengen implementieren viele Operationen sehr effizient.
- Sie sind schnell darin, zu prüfen ob ein Element in der Kollektion ist.
- Dadurch werden auch Operationen schneller, die solche Look-Ups machen müssen, z. B. Mengenoperationen.

# Zusammenfassung



- Mengen implementieren viele Operationen sehr effizient.
- Sie sind schnell darin, zu prüfen ob ein Element in der Kollektion ist.
- Dadurch werden auch Operationen schneller, die solche Look-Ups machen müssen, z. B. Mengenoperationen.
- Dafür sind Mengen aber auch ungeordnet und können jedes Element höchstens einmal beinhalten.

# Zusammenfassung



- Mengen implementieren viele Operationen sehr effizient.
- Sie sind schnell darin, zu prüfen ob ein Element in der Kollektion ist.
- Dadurch werden auch Operationen schneller, die solche Look-Ups machen müssen, z. B. Mengenoperationen.
- Dafür sind Mengen aber auch ungeordnet und können jedes Element höchstens einmal beinhalten.
- Während viele Operationen wie z. B. das Hinzufügen von Elementen zwar in  $\mathcal{O}(1)$  sind, sind sie auf Grund der technischen Realisierung manchmal doch etwas langsamer als vergleichbare Operationen von Listen.

# Zusammenfassung



- Mengen implementieren viele Operationen sehr effizient.
- Sie sind schnell darin, zu prüfen ob ein Element in der Kollektion ist.
- Dadurch werden auch Operationen schneller, die solche Look-Ups machen müssen, z. B. Mengenoperationen.
- Dafür sind Mengen aber auch ungeordnet und können jedes Element höchstens einmal beinhalten.
- Während viele Operationen wie z. B. das Hinzufügen von Elementen zwar in  $\mathcal{O}(1)$  sind, sind sie auf Grund der technischen Realisierung manchmal doch etwas langsamer als vergleichbare Operationen von Listen.
- Das Suchen bestimmter Elemente ist bei Mengen aber schneller (außer die Menge/Liste ist sehr klein...).



谢谢您们！  
Thank you!  
Vielen Dank!



# References I



- [1] AndrewBadr. "TimeComplexity". In: *The Python Wiki*. Beaverton, OR, USA: Python Software Foundation (PSF), 19. Jan. 2023. URL: <https://wiki.python.org/moin/TimeComplexity> (besucht am 2024-08-27) (siehe S. 17–25).
- [2] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume I: Introduction to SQL Queries*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-4-0. See also<sup>3</sup> (siehe S. 55, 65).
- [3] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume II: In-Depth Querying*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-5-7. See also<sup>2</sup> (siehe S. 55, 65).
- [4] Paul Gustav Heinrich Bachmann. *Die Analytische Zahlentheorie / Dargestellt von Paul Bachmann*. Bd. Zweiter Theil der Reihe Zahlentheorie: Versuch einer Gesamtdarstellung dieser Wissenschaft in ihren Haupttheilen. Leipzig, Sachsen, Germany: B. G. Teubner, 1894. ISBN: 978-1-4181-6963-3. URL: <http://gallica.bnf.fr/ark:/12148/bpt6k994750> (besucht am 2023-12-13) (siehe S. 68).
- [5] 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. 65, 67).
- [6] Daniel Bartholomew. *Learning the MariaDB Ecosystem: Enterprise-level Features for Scalability and Availability*. New York, NY, USA: Apress Media, LLC, Okt. 2019. ISBN: 978-1-4842-5514-8 (siehe S. 65).
- [7] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) und Newsgroups: alt.hypertext, 6. Aug. 1991. URL: <https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt> (besucht am 2025-02-05) (siehe S. 67).
- [8] Alex Berson. *Client/Server Architecture*. 2. Aufl. Computer Communications Series. New York, NY, USA: McGraw-Hill, 29. März 1996. ISBN: 978-0-07-005664-0 (siehe S. 64).
- [9] Silvia Botros und Jeremy Tinley. *High Performance MySQL*. 4. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Nov. 2021. ISBN: 978-1-4920-8051-0 (siehe S. 66).
- [10] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 66).

# References II



- [11] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 64).
- [12] Florian Bruhin. *Python f-Strings*. Winterthur, Switzerland: Bruhin Software, 31. Mai 2023. URL: <https://fstring.help> (besucht am 2024-07-25) (siehe S. 65).
- [13] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 65, 66).
- [14] 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](https://doi.org/10.1145/3649887). URL: <https://cacm.acm.org/research/50-years-of-queries> (besucht am 2025-01-09) (siehe S. 66).
- [15] 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. 67).
- [16] 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](https://doi.org/10.1145/362384.362685). URL: <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf> (besucht am 2025-01-05) (siehe S. 66).
- [17] Thomas H. Cormen, Charles E. Leiserson, Ronald Linn Rivest und Clifford Stein. *Introduction to Algorithms*. 3. Aufl. Cambridge, MA, USA: MIT Press, 2009. ISBN: 978-0-262-03384-8 (siehe S. 17–25).
- [18] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 66).
- [19] 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. 66).
- [20] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 66).
- [21] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebW[ How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 67).

# References III



- [22] 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. 64).
- [23] Russell J.T. Dyer. *Learning MySQL and MariaDB*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2015. ISBN: 978-1-4493-6290-4 (siehe S. 65, 66).
- [24] "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](https://docs.python.org/3/reference/lexical_analysis.html#escape-sequences) (besucht am 2025-08-05) (siehe S. 64).
- [25] 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. 66).
- [26] "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. 65).
- [27] 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. 65).
- [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. 65).
- [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. 66).
- [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. 66).

# References IV



- [31] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 65).
- [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. 65, 67).
- [33] Trey Hunner. "Python Big  $\mathcal{O}$ : The Time Complexities of Different Data Structures in Python; Python 3.8-3.12". In: *Python Morsels*. Reykjavík, Iceland: Python Morsels, 16. Apr. 2024. URL: <https://www.pythontutorial.net/time-complexities> (besucht am 2024-08-27) (siehe S. 17–25).
- [34] John Hunt. *A Beginner's Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 66).
- [35] *Information Technology – Database Languages – SQL – Part 1: Framework (SQL/Framework), Part 1*. International Standard ISO/IEC 9075-1:2023(E), Sixth Edition, (ANSI X3.135). Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Juni 2023. URL: [https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO\\_IEC\\_9075-1\\_2023\\_ed\\_6\\_-\\_id\\_76583\\_Publication\\_PDF\\_\(en\).zip](https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_(en).zip) (besucht am 2025-01-08). Consists of several parts, see <https://modern-sql.com/standard> for information where to obtain them. (Siehe S. 66).
- [36] *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. 67).
- [37] Donald Ervin Knuth. "Big Omicron and Big Omega and Big Theta". *ACM SIGACT News* 8(2):18–24, Apr.–Juni 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0163-5700. doi:10.1145/1008328.1008329 (siehe S. 68).
- [38] Donald Ervin Knuth. *Fundamental Algorithms*. 3. Aufl. Bd. 1 der Reihe *The Art of Computer Programming*. Reading, MA, USA: Addison-Wesley Professional, 1997. ISBN: 978-0-201-89683-1 (siehe S. 68).
- [39] Donald Ervin Knuth. *Sorting and Searching*. Bd. 3 der Reihe *The Art of Computer Programming*. Reading, MA, USA: Addison-Wesley Professional, 1998. ISBN: 978-0-201-89685-5 (siehe S. 17–25).

# References V



- [40] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 66).
- [41] Edmund Landau. *Handbuch der Lehre von der Verteilung der Primzahlen*. Leipzig, Sachsen, Germany: B. G. Teubner, 1909. ISBN: 978-0-8218-2650-8 (siehe S. 68).
- [42] Łukasz Langa. *Literature Overview for Type Hints*. Python Enhancement Proposal (PEP) 482. Beaverton, OR, USA: Python Software Foundation (PSF), 8. Jan. 2015. URL: <https://peps.python.org/pep-0482> (besucht am 2024-10-09) (siehe S. 67).
- [43] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 66).
- [44] Michael Lee, Ivan Levkivskyi und Jukka Lehtosalo. *Literal Types*. Python Enhancement Proposal (PEP) 586. Beaverton, OR, USA: Python Software Foundation (PSF), 14. März 2019. URL: <https://peps.python.org/pep-0586> (besucht am 2024-12-17) (siehe S. 65).
- [45] Jukka Lehtosalo, Ivan Levkivskyi, Jared Hance, Ethan Smith, Guido van Rossum, Jelle „JelleZijlstra“ Zijlstra, Michael J. Sullivan, Shantanu Jain, Xuanda Yang, Jingchen Ye, Nikita Sobolev und Mypy Contributors. *Mypy – Static Typing for Python*. San Francisco, CA, USA: GitHub Inc, 2024. URL: <https://github.com/python/mypy> (besucht am 2024-08-17) (siehe S. 66).
- [46] Gloria Lotha, Aakanksha Gaur, Erik Gregersen, Swati Chopra und William L. Hosch. "Client-Server Architecture". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 3. Jan. 2025. URL: <https://www.britannica.com/technology/client-server-architecture> (besucht am 2025-01-20) (siehe S. 64).
- [47] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 66).
- [48] MariaDB Server Documentation. Milpitas, CA, USA: MariaDB, 2025. URL: <https://mariadb.com/kb/en/documentation> (besucht am 2025-04-24) (siehe S. 65).
- [49] 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. 65).

# References VI



- [50] Jim Melton und Alan R. Simon. *SQL: 1999 – Understanding Relational Language Components*. The Morgan Kaufmann Series in Data Management Systems. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juni 2001. ISBN: 978-1-55860-456-8 (siehe S. 66).
- [51] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 64).
- [52] nishkarsh146. *Complexity Cheat Sheet for Python Operations*. Noida, Uttar Pradesh, India: GeeksforGeeks – Sanchhaya Education Private Limited, 17. Aug. 2022. URL: <https://www.geeksforgeeks.org/complexity-cheat-sheet-for-python-operations> (besucht am 2024-08-27) (siehe S. 17–25).
- [53] 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. 66).
- [54] Robert Orfali, Dan Harkey und Jeri Edwards. *Client/Server Survival Guide*. 3. Aufl. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 25. Jan. 1999. ISBN: 978-0-471-31615-2 (siehe S. 64).
- [55] Yasset Pérez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J. Eglen, Daniel S. Katz, Tom J. Pollard, Alexander Konovalov, Robert M. Flight, Kai Blin und Juan Antonio Vizcaíno. "Ten Simple Rules for Taking Advantage of Git and GitHub". *PLOS Computational Biology* 12(7), 14. Juli 2016. San Francisco, CA, USA: Public Library of Science (PLOS). ISSN: 1553-7358. doi:10.1371/JOURNAL.PCBI.1004947 (siehe S. 65).
- [56] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 66).
- [57] *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. 64).
- [58] Abhishek Ratan, Eric Chou, Pradeeban Kathiravelu und Dr. M.O. Faruque Sarker. *Python Network Programming*. Birmingham, England, UK: Packt Publishing Ltd, Jan. 2019. ISBN: 978-1-78883-546-6 (siehe S. 64).

# References VII



- [59] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: **978-1-78398-154-0** (siehe S. 65).
- [60] Mike Reichardt, Michael Gundall und Hans D. Schotten. "Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients". In: *47th Annual Conference of the IEEE Industrial Electronics Society (IECON'2021)*. 13.–15. Okt. 2021, Toronto, ON, Canada. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2021, S. 1–8. ISSN: **2577-1647**. ISBN: **978-1-6654-3554-3**. doi:[10.1109/IECON48115.2021.9589382](https://doi.org/10.1109/IECON48115.2021.9589382) (siehe S. 66).
- [61] Mark Richards und Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, Inc., Jan. 2020. ISBN: **978-1-4920-4345-4** (siehe S. 64).
- [62] "Set Types – `set`, `frozenset`". 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#set> (besucht am 2024-08-27) (siehe S. 14–16).
- [63] 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. 65).
- [64] Abraham „Avi“ Silberschatz, Henry F. „Hank“ Korth und S. Sudarshan. *Database System Concepts*. 7. Aufl. New York, NY, USA: McGraw-Hill, März 2019. ISBN: **978-0-07-802215-9** (siehe S. 17–25).
- [65] Anna Skoulakari. *Learning Git*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mai 2023. ISBN: **978-1-0981-3391-7** (siehe S. 65).
- [66] 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. 65).
- [67] 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](https://doi.org/10.1145/361020.361025) (siehe S. 66).
- [68] "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. 66).

# References VIII



- [69] Ryan K. Stephens und Ronald R. Plew. *Sams Teach Yourself SQL in 21 Days*. 4. Aufl. Sams Tech Yourself. Indianapolis, IN, USA: SAMS Technical Publishing und Hoboken, NJ, USA: Pearson Education, Inc., Okt. 2002. ISBN: 978-0-672-32451-2 (siehe S. 62, 66).
- [70] 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<sup>69</sup> (siehe S. 66).
- [71] "String Constants". In: Kap. 4.1.2.1. URL: <https://www.postgresql.org/docs/17/sql-syntax-lexical.html#SQL-SYNTAX-STRINGS> (besucht am 2025-08-23) (siehe S. 64).
- [72] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 66).
- [73] 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. 66).
- [74] ."Literals". In: *Static Typing with Python*. Hrsg. von The Python Typing Team. Beaverton, OR, USA: Python Software Foundation (PSF), 2021. URL: <https://typing.python.org/en/latest/spec/literal.html> (besucht am 2025-08-29) (siehe S. 65).
- [75] *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. 67).
- [76] 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. 65).
- [77] Mariot Tsitoara. *Beginning Git and GitHub: Version Control, Project Management and Teamwork for the New Developer*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0215-7 (siehe S. 65, 67).
- [78] *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. 67).

# References IX



- [79] Guido van Rossum und Łukasz Langa. *Type Hints*. Python Enhancement Proposal (PEP) 484. Beaverton, OR, USA: Python Software Foundation (PSF), 29. Sep. 2014. URL: <https://peps.python.org/pep-0484> (besucht am 2024-08-22) (siehe S. 67).
- [80] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 65).
- [81] Thomas Weise (汤卫思). *Databases*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2025. URL: <https://thomasweise.github.io/databases> (besucht am 2025-01-05) (siehe S. 64, 66).
- [82] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: <https://thomasweise.github.io/programmingWithPython> (besucht am 2025-01-05) (siehe S. 64, 66).
- [83] *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. 66).
- [84] Ulf Michael „Monty“ Widenius, David Axmark und Uppsala, Sweden: MySQL AB. *MySQL Reference Manual – Documentation from the Source*. Sebastopol, CA, USA: O'Reilly Media, Inc., 9. Juli 2002. ISBN: 978-0-596-00265-7 (siehe S. 66).
- [85] 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. 64).
- [86] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 64).

# Glossary (in English) I



**Bash** is the shell used under Ubuntu Linux, i.e., the program that „runs“ in the terminal and interprets your commands, allowing you to start and interact with other programs<sup>11,51,86</sup>. Learn more at <https://www.gnu.org/software/bash>.

**C** is a programming language, which is very successful in system programming situations<sup>22,57</sup>.

**client** In a client-server architecture, the client is a device or process that requests a service from the server. It initiates the communication with the server, sends a request, and receives the response with the result of the request. Typical examples for clients are web browsers in the internet as well as clients for database management systems (DBMSes), such as psql.

**client-server architecture** is a system design where a central server receives requests from one or multiple clients<sup>8,46,54,58,61</sup>. These requests and responses are usually sent over network connections. A typical example for such a system is the World Wide Web (WWW), where web servers host websites and make them available to web browsers, the clients. Another typical example is the structure of database (DB) software, where a central server, the DBMS, offers access to the DB to the different clients. Here, the client can be some terminal software shipping with the DBMS, such as psql, or the different applications that access the DBs.

**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*<sup>81</sup>.

**DBMS** A *database management system* is the software layer located between the user or application and the DB. The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB<sup>85</sup>.

**escape sequence** Escaping is the process of presenting „forbidden“ characters or symbols in a sequence of characters or symbols. In Python<sup>82</sup>, string escapes allow us to include otherwise impossible characters, such as string delimiters, in a string. Each such character is represented by an *escape sequence*, which usually starts with the backslash character („\“)<sup>24</sup>. In Python strings, the escape sequence „\“ , for example, stands for „\“, the escape sequence „\\“ stands for „\“, and the escape sequence „\n“ stands for a newline or linebreak character. In Python f-strings, the escape sequence „{{“ stands for a single curly brace { . In PostgreSQL<sup>81</sup>, similar C-style escapes (starting with „\“) are supported<sup>71</sup>.

# Glossary (in English) II



**f-string** let you include the results of expressions in strings<sup>12,26–28,49,66</sup>. They can contain expressions (in curly braces) like `f"{}{}` that are then transformed to text via (string) interpolation, which turns the string to `"a5b"`. F-strings are delimited by `f"..."`.

**Git** is a distributed Version Control Systems (VCS) which allows multiple users to work on the same code while preserving the history of the code changes<sup>65,77</sup>. Learn more at <https://git-scm.com>.

**GitHub** is a website where software projects can be hosted and managed via the Git VCS<sup>55,77</sup>. Learn more at <https://github.com>.

**IT** information technology

**LAMP Stack** A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP<sup>13,32</sup>.

**Linux** is the leading open source operating system, i.e., a free alternative for Microsoft Windows<sup>5,31,63,76,80</sup>. 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.

**literal** A literal is a specific concrete value, something that is written down as-is<sup>44,74</sup>. In Python, for example, `"abc"` is a string literal, `5` is an integer literal, and `23.3` is a `float` literal. In contrast, `sin(3)` is not a literal. Also, while `5` is an integer literal, if we create a variable `a = 5` then `a` is not a literal either (it is a variable). Hence, literals are values that the Python interpreter reads directly from the source code and creates as objects in memory. They are not something that is the result from a computation or the result of a variable lookup. Python supports some type hints for literals, including the type `LiteralString` for string literals and the type `Literal[xyz]` for arbitrary literals `xyz`.

**MariaDB** An open source relational database management system that has forked off from MySQL<sup>2,3,6,23,48,59</sup>. See <https://mariadb.org> for more information.

# Glossary (in English) III



**Microsoft Windows** is a commercial proprietary operating system<sup>10</sup>. 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>.

**Mypy** is a static type checking tool for Python<sup>45</sup> that makes use of type hints. Learn more at <https://github.com/python/mypy> and in<sup>82</sup>.

**MySQL** An open source relational database management system<sup>9,23,60,73,84</sup>. MySQL is famous for its use in the LAMP Stack. See <https://www.mysql.com> for more information.

**PostgreSQL** An open source object-relational DBMS<sup>25,53,56,73</sup>. See <https://postgresql.org> for more information.

**psql** is the client program used to access the PostgreSQL DBMS server.

**Python** The Python programming language<sup>34,43,47,82</sup>, i.e., what you will learn about in our book<sup>82</sup>. 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<sup>16,29,30,67,72,81,83</sup>.

**server** In a client-server architecture, the server is a process that fulfills the requests of the clients. It usually waits for incoming communication carrying the requests from the clients. For each request, it takes the necessary actions, performs the required computations, and then sends a response with the result of the request. Typical examples for servers are web servers<sup>13</sup> in the internet as well as DBMSes. It is also common to refer to the computer running the server processes as server as well, i.e., to call it the „server computer“<sup>40</sup>.

**SQL** The *Structured Query Language* is basically a programming language for querying and manipulating relational databases<sup>14,18–20,35,50,68–70,72</sup>. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference<sup>68</sup>.

# Glossary (in English) IV



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

**terminal** A terminal is a text-based window where you can enter commands and execute them<sup>5,15</sup>. Knowing what a terminal is and how to use it is very essential in any programming- or system administration-related task. If you want to open a terminal under Microsoft Windows, you can Druck auf + , dann Schreiben von `cmd`, dann Druck auf . Under Ubuntu Linux, + + opens a terminal, which then runs a Bash shell inside.

**type hint** are annotations that help programmers and static code analysis tools such as Mypy to better understand what type a variable or function parameter is supposed to be<sup>42,79</sup>. Python is a dynamically typed programming language where you do not need to specify the type of, e.g., a variable. This creates problems for code analysis, both automated as well as manual: For example, it may not always be clear whether a variable or function parameter should be an integer or floating point number. The annotations allow us to explicitly state which type is expected. They are *ignored* during the program execution. They are a basically a piece of documentation.

**Ubuntu** is a variant of the open source operating system Linux<sup>15,32</sup>. 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<sup>36,75,78</sup>. 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.

**VCS** A *Version Control System* is a software which allows you to manage and preserve the historical development of your program code<sup>77</sup>. A distributed VCS allows multiple users to work on the same code and upload their changes to the server, which then preserves the change history. The most popular distributed VCS is Git.

**WWW** World Wide Web<sup>7,21</sup>



# Glossary (in English) V

- $\Omega(g(x))$  If  $f(x) = \Omega(g(x))$ , then there exist positive numbers  $x_0 \in \mathbb{R}^+$  and  $c \in \mathbb{R}^+$  such that  $f(x) \geq c * g(x) \geq 0 \forall x \geq x_0^{37,38}$ . In other words,  $\Omega(g(x))$  describes a lower bound for function growth.
- $\mathcal{O}(g(x))$  If  $f(x) = \mathcal{O}(g(x))$ , then there exist positive numbers  $x_0 \in \mathbb{R}^+$  and  $c \in \mathbb{R}^+$  such that  $0 \leq f(x) \leq c * g(x) \forall x \geq x_0^{4,37,38,41}$ . In other words,  $\mathcal{O}(g(x))$  describes an upper bound for function growth.
- $\Theta(g(x))$  If  $f(x) = \Theta(g(x))$ , then  $f(x) = \mathcal{O}(g(x))$  and  $f(x) = \Omega(g(x))^{37,38}$ . In other words,  $\Theta(g(x))$  describes an exact order of function growth.

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

$\mathbb{R}^+$  the set of the positive real numbers, i.e.,  $\mathbb{R}^+ = \{x \in \mathbb{R} : x > 0\}$ .