合肥大學
HEFEI UNIVERSITY

Programming with Python

37. Set und Dictionary Comprehension

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

Institute of Applied Optimization (IAO) | 应用优化研究所
School of Artificial Intelligence and Big Data | 人工智能与大数据学院
Hefei University | 合肥大学
Hefei, Anhui, China | 中国安徽省合肥市

Version: 2025-10-30

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

Einleitung

Comprehension: Nur für Listen?

- Wir haben bereits List Comprehension gelernt.

Comprehension: Nur für Listen?

- Wir haben bereits List Comprehension gelernt.
- Das ist ein sehr elegantes und mächtiges Werkzeug um, nun ja, Listen zu erstellen.

Comprehension: Nur für Listen?

- Wir haben bereits List Comprehension gelernt.
- Das ist ein sehr elegantes und mächtiges Werkzeug um, nun ja, Listen zu erstellen.
- Wäre es nicht eigenartig wenn so ein Werkzeug *nur* für Listen zur Verfügung stünde?

Comprehension: Nur für Listen?

- Wir haben bereits List Comprehension gelernt.
- Das ist ein sehr elegantes und mächtiges Werkzeug um, nun ja, Listen zu erstellen.
- Wäre es nicht eigenartig wenn so ein Werkzeug *nur* für Listen zur Verfügung stünde?
- Was ist mit den anderen Kollektions-Datentypen?

Comprehension: Nur für Listen?

- Wir haben bereits List Comprehension gelernt.
- Das ist ein sehr elegantes und mächtiges Werkzeug um, nun ja, Listen zu erstellen.
- Wäre es nicht eigenartig wenn so ein Werkzeug *nur* für Listen zur Verfügung stünde?
- Was ist mit den anderen Kollektions-Datentypen?
- Tatsächlich gibt es das auch für Mengen und Dictionaries.

Comprehension: Nur für Listen?

- Wir haben bereits List Comprehension gelernt.
- Das ist ein sehr elegantes und mächtiges Werkzeug um, nun ja, Listen zu erstellen.
- Wäre es nicht eigenartig wenn so ein Werkzeug *nur* für Listen zur Verfügung stünde?
- Was ist mit den anderen Kollektions-Datentypen?
- Tatsächlich gibt es das auch für Mengen und Dictionaries.
- Wir schauen uns jetzt Ersteres an und danach dann Letzteres…

Set Comprehension

Set Comprehension

```python
"""Set Comprehension in Python."""

# Create a set from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
{expression for item in sequence}

# Create a set from those items in a sequence for which 'condition'
# evaluates to True.
# 'expression' and 'condition' are usually expressions whose results
# depend on 'item'.
{expression for item in sequence if condition}
```

- Mengen können mit Set Comprehension erstellt werden.

Set Comprehension

```python
"""Set Comprehension in Python."""

# Create a set from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
{expression for item in sequence}

# Create a set from those items in a sequence for which 'condition'
# evaluates to True.
# 'expression' and 'condition' are usually expressions whose results
# depend on 'item'.
{expression for item in sequence if condition}
```

- Mengen können mit Set Comprehension erstellt werden.
- Set Comprehension funktioniert genau so wie List Comprehension.

Set Comprehension

```python
"""Set Comprehension in Python."""

# Create a set from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
{expression for item in sequence}

# Create a set from those items in a sequence for which 'condition'
# evaluates to True.
# 'expression' and 'condition' are usually expressions whose results
# depend on 'item'.
{expression for item in sequence if condition}
```

- Mengen können mit Set Comprehension erstellt werden.
- Set Comprehension funktioniert genau so wie List Comprehension.
- Die Syntax ist im Grunde gleich und unterscheidet sich nur in der Verwendung von geschweiften Klammern an Stelle von eckigen Klammern.

Beispiel

- Schauen wir uns ein paar Beispiele als Programm `simple_set_comprehension.py` an.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Schauen wir uns ein paar Beispiele als Programm `simple_set_comprehension.py` an.
- In dem Programm erstellen wir erst eine Menge mit den Ergebnissen der `isqrt`-Funktion aus dem Modul `math`.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Schauen wir uns ein paar Beispiele als Programm `simple_set_comprehension.py` an.
- In dem Programm erstellen wir erst eine Menge mit den Ergebnissen der `isqrt`-Funktion aus dem Modul `math`.
- Diese Funktion liefert den ganzzahligen Teil der Quadratwurzel, also `isqrt(i)` $= \lfloor\sqrt{\texttt{i}}\rfloor$.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Schauen wir uns ein paar Beispiele als Programm `simple_set_comprehension.py` an.
- In dem Programm erstellen wir erst eine Menge mit den Ergebnissen der `isqrt`-Funktion aus dem Modul `math`.
- Diese Funktion liefert den ganzzahligen Teil der Quadratwurzel, also $\texttt{isqrt(i)} = \lfloor\sqrt{\texttt{i}}\rfloor$.
- Wir wollen eine Menge erstellen mit allen Ergebnissen dieser Funktion für die Werte `i` von 0 bis 99.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Schauen wir uns ein paar Beispiele als Programm `simple_set_comprehension.py` an.
- In dem Programm erstellen wir erst eine Menge mit den Ergebnissen der `isqrt`-Funktion aus dem Modul `math`.
- Diese Funktion liefert den ganzzahligen Teil der Quadratwurzel, also `isqrt(i)` $= \lfloor\sqrt{\texttt{i}}\rfloor$.
- Wir wollen eine Menge erstellen mit allen Ergebnissen dieser Funktion für die Werte `i` von 0 bis 99.
- Wir probieren zwei Möglichkeiten, um so eine Menge zu erstellen.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- In dem Programm erstellen wir erst eine Menge mit den Ergebnissen der `isqrt`-Funktion aus dem Modul `math`.
- Diese Funktion liefert den ganzzahligen Teil der Quadratwurzel, also `isqrt(i)` $= \lfloor\sqrt{\texttt{i}}\rfloor$.
- Wir wollen eine Menge erstellen mit allen Ergebnissen dieser Funktion für die Werte `i` von 0 bis 99.
- Wir probieren zwei Möglichkeiten, um so eine Menge zu erstellen.
- Wir erstellen zuerst eine leere Menge `roots_1`.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Diese Funktion liefert den ganzzahligen Teil der Quadratwurzel, also `isqrt(i)` $= \lfloor\sqrt{\texttt{i}}\rfloor$.
- Wir wollen eine Menge erstellen mit allen Ergebnissen dieser Funktion für die Werte `i` von 0 bis 99.
- Wir probieren zwei Möglichkeiten, um so eine Menge zu erstellen.
- Wir erstellen zuerst eine leere Menge `roots_1`.
- Dann fügen wir iterativ neue Ergebnisse von `isqrt` mit Hilfe von `roots_1.add` in einer `for`-Schleife an.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Wir wollen eine Menge erstellen mit allen Ergebnissen dieser Funktion für die Werte `i` von 0 bis 99.
- Wir probieren zwei Möglichkeiten, um so eine Menge zu erstellen.
- Wir erstellen zuerst eine leere Menge `roots_1`.
- Dann fügen wir iterativ neue Ergebnisse von `isqrt` mit Hilfe von `roots_1.add` in einer `for`-Schleife an.
- Die Menge enthält danach die Zahlen von 0 bis 9.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Wir probieren zwei Möglichkeiten, um so eine Menge zu erstellen.
- Wir erstellen zuerst eine leere Menge `roots_1`.
- Dann fügen wir iterativ neue Ergebnisse von `isqrt` mit Hilfe von `roots_1.add` in einer `for`-Schleife an.
- Die Menge enthält danach die Zahlen von 0 bis 9.
- Jeder Wert taucht genau einmal auf, denn so funktionieren Mengen nunmal.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Wir erstellen zuerst eine leere Menge `roots_1`.
- Dann fügen wir iterativ neue Ergebnisse von `isqrt` mit Hilfe von `roots_1.add` in einer `for`-Schleife an.
- Die Menge enthält danach die Zahlen von 0 bis 9.
- Jeder Wert taucht genau einmal auf, denn so funktionieren Mengen nunmal.
- Wir können die selbe Menge mit Set Comprehension erstellen.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Dann fügen wir iterativ neue Ergebnisse von `isqrt` mit Hilfe von `roots_1.add` in einer `for`-Schleife an.
- Die Menge enthält danach die Zahlen von 0 bis 9.
- Jeder Wert taucht genau einmal auf, denn so funktionieren Mengen nunmal.
- Wir können die selbe Menge mit Set Comprehension erstellen.
- Die Menge `roots_2` wird durch `{isqrt(j)for j in range(100)}` erstellt.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Die Menge enthält danach die Zahlen von 0 bis 9.
- Jeder Wert taucht genau einmal auf, denn so funktionieren Mengen nunmal.
- Wir können die selbe Menge mit Set Comprehension erstellen.
- Die Menge `roots_2` wird durch `{isqrt(j)for j in range(100)}` erstellt.
- Sie ist genau gleich zu `roots_1`.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Jeder Wert taucht genau einmal auf, denn so funktionieren Mengen nunmal.
- Wir können die selbe Menge mit Set Comprehension erstellen.
- Die Menge `roots_2` wird durch `{isqrt(j)for j in range(100)}` erstellt.
- Sie ist genau gleich zu `roots_1`.
- Set Comprehension funktioniert genau wie List Comprehension.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Wir können die selbe Menge mit Set Comprehension erstellen.
- Die Menge `roots_2` wird durch `{isqrt(j)for j in range(100)}` erstellt.
- Sie ist genau gleich zu `roots_1`.
- Set Comprehension funktioniert genau wie List Comprehension.
- Unsere ersten Beispiele sind auch ziemlich ähnlich.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Die Menge `roots_2` wird durch `{isqrt(j)for j in range(100)}` erstellt.
- Sie ist genau gleich zu `roots_1`.
- Set Comprehension funktioniert genau wie List Comprehension.
- Unsere ersten Beispiele sind auch ziemlich ähnlich.
- Machen wir nun etwas Interessanteres.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Sie ist genau gleich zu `roots_1`.
- Set Comprehension funktioniert genau wie List Comprehension.
- Unsere ersten Beispiele sind auch ziemlich ähnlich.
- Machen wir nun etwas Interessanteres.
- Wir wollen die Menge `primes` der Primzahlen[1,5,8] aus 2..99 erstellen.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Set Comprehension funktioniert genau wie List Comprehension.
- Unsere ersten Beispiele sind auch ziemlich ähnlich.
- Machen wir nun etwas Interessanteres.
- Wir wollen die Menge `primes` der Primzahlen[1,5,8] aus 2..99 erstellen.
- Wir hatten schon mal ein Program, das das macht, in Einheit 23.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Unsere ersten Beispiele sind auch ziemlich ähnlich.
- Machen wir nun etwas Interessanteres.
- Wir wollen die Menge `primes` der Primzahlen[1,5,8] aus 2..99 erstellen.
- Wir hatten schon mal ein Program, das das macht, in Einheit 23.
- Dieses mal wollen wir Set Comprehension verwenden.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Machen wir nun etwas Interessanteres.
- Wir wollen die Menge `primes` der Primzahlen[1,5,8] aus 2..99 erstellen.
- Wir hatten schon mal ein Program, das das macht, in Einheit 23.
- Dieses mal wollen wir Set Comprehension verwenden.
- Wir erstellen dafür erstmal eine Menge `not_primes` mit Zahlen, die *keine* Primzahlen sind.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Wir wollen die Menge `primes` der Primzahlen[1,5,8] aus 2..99 erstellen.
- Wir hatten schon mal ein Program, das das macht, in Einheit 23.
- Dieses mal wollen wir Set Comprehension verwenden.
- Wir erstellen dafür erstmal eine Menge `not_primes` mit Zahlen, die *keine* Primzahlen sind.
- Wir lassen dafür die Variable `k` von `2` bis `99` iterieren.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Wir hatten schon mal ein Program, das das macht, in Einheit 23.
- Dieses mal wollen wir Set Comprehension verwenden.
- Wir erstellen dafür erstmal eine Menge `not_primes` mit Zahlen, die *keine* Primzahlen sind.
- Wir lassen dafür die Variable `k` von `2` bis `99` iterieren.
- Für jede Zahl `k` lassen wir eine zweite Variable `m` von `2` bis `isqrt(k)` iterieren.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Dieses mal wollen wir Set Comprehension verwenden.
- Wir erstellen dafür erstmal eine Menge `not_primes` mit Zahlen, die *keine* Primzahlen sind.
- Wir lassen dafür die Variable `k` von `2` bis `99` iterieren.
- Für jede Zahl `k` lassen wir eine zweite Variable `m` von `2` bis `isqrt(k)` iterieren.
- Für jede einzelne resultierende `k`-`m` Kombination fügen wir den Wert `k` in die Menge ein wenn die Bedingung `k % m == 0` zutrifft.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Wir erstellen dafür erstmal eine Menge `not_primes` mit Zahlen, die *keine* Primzahlen sind.
- Wir lassen dafür die Variable `k` von `2` bis `99` iterieren.
- Für jede Zahl `k` lassen wir eine zweite Variable `m` von `2` bis `isqrt(k)` iterieren.
- Für jede einzelne resultierende `k`-`m` Kombination fügen wir den Wert `k` in die Menge ein wenn die Bedingung `k % m == 0` zutrifft.
- Jedesmal wenn wir eine Zahl `m` finden, die `k` ohne Rest teilt, dann fügen wir `k` in die Menge ein.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Wir lassen dafür die Variable `k` von `2` bis `99` iterieren.
- Für jede Zahl `k` lassen wir eine zweite Variable `m` von `2` bis `isqrt(k)` iterieren.
- Für jede einzelne resultierende `k`-`m` Kombination fügen wir den Wert `k` in die Menge ein **wenn** die Bedingung `k % m == 0` zutrifft.
- Jedesmal wenn wir eine Zahl `m` finden, die `k` ohne Rest teilt, dann fügen wir `k` in die Menge ein.
- Würden wir List Comprehension machen, dann würden wir eine riesige Liste bekommen, in der viele Werte von `k` wiederholt auftreten.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Für jede einzelne resultierende `k`-`m` Kombination fügen wir den Wert `k` in die Menge ein wenn die Bedingung `k % m == 0` zutrifft.
- Jedesmal wenn wir eine Zahl `m` finden, die `k` ohne Rest teilt, dann fügen wir `k` in die Menge ein.
- Würden wir List Comprehension machen, dann würden wir eine riesige Liste bekommen, in der viele Werte von `k` wiederholt auftreten.
- 96, z. B., würde fünf Mal auftauchen, weil es durch 2, 3, 4, 6, und 8 teilbar ist, die all weniger als $9 = \lfloor\sqrt{96}\rfloor$ sind.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Jedesmal wenn wir eine Zahl `m` finden, die `k` ohne Rest teilt, dann fügen wir `k` in die Menge ein.
- Würden wir List Comprehension machen, dann würden wir eine riesige Liste bekommen, in der viele Werte von `k` wiederholt auftreten.
- 96, z. B., würde fünf Mal auftauchen, weil es durch 2, 3, 4, 6, und 8 teilbar ist, die all weniger als $9 = \lfloor\sqrt{96}\rfloor$ sind.
- Wir machen allerdings Set Comprehension, also bekommen wir jeden Wert höchstens einmal.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Jedesmal wenn wir eine Zahl `m` finden, die `k` ohne Rest teilt, dann fügen wir `k` in die Menge ein.
- Würden wir List Comprehension machen, dann würden wir eine riesige Liste bekommen, in der viele Werte von `k` wiederholt auftreten.
- 96, z. B., würde fünf Mal auftauchen, weil es durch 2, 3, 4, 6, und 8 teilbar ist, die all weniger als $9 = \lfloor\sqrt{96}\rfloor$ sind.
- Wir machen allerdings Set Comprehension, also bekommen wir jeden Wert höchstens einmal.
- Diese extrem ineffiziente Methode gibt uns die Menge `not_primes`.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- 96, z. B., würde fünf Mal auftauchen, weil es durch 2, 3, 4, 6, und 8 teilbar ist, die all weniger als $9 = \lfloor\sqrt{96}\rfloor$ sind.
- Wir machen allerdings Set Comprehension, also bekommen wir jeden Wert höchstens einmal.
- Diese extrem ineffiziente Methode gibt uns die Menge `not_primes`.
- Wenn eine Zahl durch eine andere Zahl geteilt werden kann, die größer als 1 und kleiner als die Zahl selber ist, dann kann die Zahl keine Primzahl sein.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Wir machen allerdings Set Comprehension, also bekommen wir jeden Wert höchstens einmal.
- Diese extrem ineffiziente Methode gibt uns die Menge `not_primes`.
- Wenn eine Zahl durch eine andere Zahl geteilt werden kann, die größer als 1 und kleiner als die Zahl selber ist, dann kann die Zahl keine Primzahl sein.
- Mit Hilfe der Zahlen die keine Primzahlen sind, können wir nun Set Comprehension anwenden, um die Zahlen, die Primzahlen sind, zu bekommen.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Diese extrem ineffiziente Methode gibt uns die Menge `not_primes`.
- Wenn eine Zahl durch eine andere Zahl geteilt werden kann, die größer als 1 und kleiner als die Zahl selber ist, dann kann die Zahl keine Primzahl sein.
- Mit Hilfe der Zahlen die keine Primzahlen sind, können wir nun Set Comprehension anwenden, um die Zahlen, die Primzahlen sind, zu bekommen.
- `{n for n in range(2, 100) if n not in not_primes}` lässt eine Variable `n` von 2 bis 99 laufen.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Wenn eine Zahl durch eine andere Zahl geteilt werden kann, die größer als 1 und kleiner als die Zahl selber ist, dann kann die Zahl keine Primzahl sein.
- Mit Hilfe der Zahlen die keine Primzahlen sind, können wir nun Set Comprehension anwenden, um die Zahlen, die Primzahlen sind, zu bekommen.
- `{n for n in range(2, 100) if n not in not_primes}` lässt eine Variable `n` von 2 bis 99 laufen.
- Es behält dann jeden Wert von `n` in der Menge wenn `n` nicht schon in der Menge `not_primes` ist.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Mit Hilfe der Zahlen die keine Primzahlen sind, können wir nun Set Comprehension anwenden, um die Zahlen, die Primzahlen sind, zu bekommen.
- `{n for n in range(2, 100) if n not in not_primes}` lässt eine Variable `n` von 2 bis 99 laufen.
- Es behält dann jeden Wert von `n` in der Menge wenn `n` nicht schon in der Menge `not_primes` ist.
- So bekommen wir die Menge der Primzahlen korrekt, auch wenn unsere Methode insgesamt ineffizient ist.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- `{n for n in range(2, 100) if n not in not_primes}` lässt eine Variable `n` von 2 bis 99 laufen.
- Es behält dann jeden Wert von `n` in der Menge wenn `n` nicht schon in der Menge `not_primes` ist.
- So bekommen wir die Menge der Primzahlen korrekt, auch wenn unsere Methode insgesamt ineffizient ist.
- Als kleine Auffrischung erinnern wir uns nochmal kurz an die Mengenoperationen aus Einheit 20 und daran, dass wir eine Menge auch erstellen können, wenn wir eine Sequenz als Parameter der Funktion `set` angeben.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- Es behält dann jeden Wert von `n` in der Menge wenn `n` nicht schon in der Menge `not_primes` ist.
- So bekommen wir die Menge der Primzahlen korrekt, auch wenn unsere Methode insgesamt ineffizient ist.
- Als kleine Auffrischung erinnern wir uns nochmal kurz an die Mengenoperationen aus Einheit 20 und daran, dass wir eine Menge auch erstellen können, wenn wir eine Sequenz als Parameter der Funktion `set` angeben.
- `set(range(2, 100))` erstellt eine Menge mit allen Zahlen aus 2..99.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- So bekommen wir die Menge der Primzahlen korrekt, auch wenn unsere Methode insgesamt ineffizient ist.
- Als kleine Auffrischung erinnern wir uns nochmal kurz an die Mengenoperationen aus Einheit 20 und daran, dass wir eine Menge auch erstellen können, wenn wir eine Sequenz als Parameter der Funktion `set` angeben.
- `set(range(2, 100))` erstellt eine Menge mit allen Zahlen aus 2..99.
- `set(range(2, 100)).difference(not_primes)` liefert dann die Menge aller dieser Elemente die nicht auch in `not_primes` vorkommen.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ python3 simple_set_comprehension.py ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
   ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Beispiel

- `set(range(2, 100))` erstellt eine Menge mit allen Zahlen aus 2..99.
- `set(range(2, 100)).difference(not_primes)` liefert dann die Menge aller dieser Elemente die nicht auch in `not_primes` vorkommen.
- So könnte man also das Gleiche erreichen, zumindest wenn man schon die Menge `not_primes` gebaut hat.

```python
"""Simple examples for set comprehension."""

from math import isqrt  # computes the integer parts of square roots

roots_1: set[int] = set()  # We can start with an empty set.
for i in range(100):  # Then we use a for-loop over the numbers 0 to 99.
    roots_1.add(isqrt(i))  # Add the integer part of sqrt to the set.
print(f" result of construction: {roots_1}")  # Print the result.

# Or we use set comprehension as follows:
roots_2: set[int] = {isqrt(j) for j in range(100)}
print(f"result of comprehension: {roots_2}")  # Print the result.

# Compute the set of numbers in 2..99 which are not prime.
not_primes: set[int] = {k for k in range(2, 100)
                        for m in range(2, isqrt(k) + 1) if k % m == 0}
# The set of numbers in 2..99 which are not in not_primes are primes.
primes: set[int] = {n for n in range(2, 100) if n not in not_primes}
print(f"prime numbers 1: {primes}")

# We could also use this method that creates a set from a range and uses
# the set difference operator.
print(f"prime numbers 2: {set(range(2, 100)).difference(not_primes)}")
```

↓ `python3 simple_set_comprehension.py` ↓

```
 result of construction: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
result of comprehension: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
prime numbers 1: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
prime numbers 2: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    ↪ 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```

Dictionary Comprehension

Dictionary Comprehension

- Dictionary Comprehension funktioniert wieder fast genauso wie Set- und List Comprehension[6].

```python
"""Dictionary Comprehension in Python."""

# Create a dictionary from all the item pairs in a sequence.
# 'expression1' and 'expression2' are usually expressions whose results
# depend on 'item'.
{expression1: expression2 for item in sequence}

# Create a dictionary from those items in a sequence for which
# 'condition' evaluates to True.
# 'expression1', 'expression2', and 'condition' are usually expressions
# whose results depend on 'item'.
{expression1: expression2 for item in sequence if condition}
```

Dictionary Comprehension

- Dictionary Comprehension funktioniert wieder fast genauso wie Set- und List Comprehension[6].
- Anders als diese weist es aber Werten zu Schlüssel zu und braucht daher zwei Ausdrücke pro Eintrag, welche durch `:` getrennt werden.

```python
"""Dictionary Comprehension in Python."""

# Create a dictionary from all the item pairs in a sequence.
# 'expression1' and 'expression2' are usually expressions whose results
# depend on 'item'.
{expression1: expression2 for item in sequence}

# Create a dictionary from those items in a sequence for which
# 'condition' evaluates to True.
# 'expression1', 'expression2', and 'condition' are usually expressions
# whose results depend on 'item'.
{expression1: expression2 for item in sequence if condition}
```

Dictionary Comprehension

- Dictionary Comprehension funktioniert wieder fast genauso wie Set- und List Comprehension[6].
- Anders als diese weist es aber Werten zu Schlüssel zu und braucht daher zwei Ausdrücke pro Eintrag, welche durch `:` getrennt werden.
- Das ist auch der Unterschied zwischen Set- und Dictionary Comprehension.

```python
"""Dictionary Comprehension in Python."""

# Create a dictionary from all the item pairs in a sequence.
# 'expression1' and 'expression2' are usually expressions whose results
# depend on 'item'.
{expression1: expression2 for item in sequence}

# Create a dictionary from those items in a sequence for which
# 'condition' evaluates to True.
# 'expression1', 'expression2', and 'condition' are usually expressions
# whose results depend on 'item'.
{expression1: expression2 for item in sequence if condition}
```

Dictionary Comprehension

- Anders als diese weist es aber Werten zu Schlüssel zu und braucht daher zwei Ausdrücke pro Eintrag, welche durch `:` getrennt werden.
- Das ist auch der Unterschied zwischen Set- und Dictionary Comprehension.
- Beide benutzen geschwungene Klammern.

```python
"""Dictionary Comprehension in Python."""

# Create a dictionary from all the item pairs in a sequence.
# 'expression1' and 'expression2' are usually expressions whose results
# depend on 'item'.
{expression1: expression2 for item in sequence}

# Create a dictionary from those items in a sequence for which
# 'condition' evaluates to True.
# 'expression1', 'expression2', and 'condition' are usually expressions
# whose results depend on 'item'.
{expression1: expression2 for item in sequence if condition}
```

Dictionary Comprehension

- Das ist auch der Unterschied zwischen Set- und Dictionary Comprehension.
- Beide benutzen geschwungene Klammern.
- In der Dictionary Comprehension werden Schlüssel und Werte durch `:` getrennt, wobei bei Set Comprehension, nur einzelne Werte angegeben werden.

```python
"""Dictionary Comprehension in Python."""

# Create a dictionary from all the item pairs in a sequence.
# 'expression1' and 'expression2' are usually expressions whose results
# depend on 'item'.
{expression1: expression2 for item in sequence}

# Create a dictionary from those items in a sequence for which
# 'condition' evaluates to True.
# 'expression1', 'expression2', and 'condition' are usually expressions
# whose results depend on 'item'.
{expression1: expression2 for item in sequence if condition}
```

Beispiel

- Mit `simple_dict_comprehension.py` schauen wir uns ein paar Beispiele für Dictionary Comprehension an.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9: 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1, 18: 9, 19: 1, 20: 10}
```

Beispiel

- Mit simple_dict_comprehension.py schauen wir uns ein paar Beispiele für Dictionary Comprehension an.
- Wir fangen wieder damit an, ein Doctionary „manuell“ zu erstellen und werden dann sehen, dass Dictionary Comprehension kompakter ist.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"      largest divisors: {maxdiv}")
```

↓ python3 simple_dict_comprehension.py ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
      largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9: 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1, 18: 9, 19: 1, 20: 10}
```

Beispiel

- Mit `simple_dict_comprehension.py` schauen wir uns ein paar Beispiele für Dictionary Comprehension an.
- Wir fangen wieder damit an, ein Doctionary „manuell“ zu erstellen und werden dann sehen, dass Dictionary Comprehension kompakter ist.
- Wir wollen eine Datenstruktur mit den Quadraten der Zahlen von 0 bis 10 erstellen.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
   ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
   ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Mit `simple_dict_comprehension.py` schauen wir uns ein paar Beispiele für Dictionary Comprehension an.
- Wir fangen wieder damit an, ein Doctionary „manuell“ zu erstellen und werden dann sehen, dass Dictionary Comprehension kompakter ist.
- Wir wollen eine Datenstruktur mit den Quadraten der Zahlen von 0 bis 10 erstellen.
- Dieses Mal benutzen wir ein Dictionary, wo wir die Quadrate als Werte den Zahlen als Schlüsseln zuweisen.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9: 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1, 18: 9, 19: 1, 20: 10}
```

Beispiel

- Wir fangen wieder damit an, ein Doctionary „manuell“ zu erstellen und werden dann sehen, dass Dictionary Comprehension kompakter ist.
- Wir wollen eine Datenstruktur mit den Quadraten der Zahlen von 0 bis 10 erstellen.
- Dieses Mal benutzen wir ein Dictionary, wo wir die Quadrate als Werte den Zahlen als Schlüsseln zuweisen.
- Wir fangen also mit einem leeren Doctionary `squares_1` an.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9: 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1, 18: 9, 19: 1, 20: 10}
```

Beispiel

- Wir fangen wieder damit an, ein Doctionary „manuell“ zu erstellen und werden dann sehen, dass Dictionary Comprehension kompakter ist.
- Wir wollen eine Datenstruktur mit den Quadraten der Zahlen von 0 bis 10 erstellen.
- Dieses Mal benutzen wir ein Dictionary, wo wir die Quadrate als Werte den Zahlen als Schlüsseln zuweisen.
- Wir fangen also mit einem leeren Doctionary `squares_1` an.
- Dann benutzen wir eine `for`-Schleife und iterieren die Variable `i` über `range(11)`.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
   ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
   ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Wir wollen eine Datenstruktur mit den Quadraten der Zahlen von 0 bis 10 erstellen.
- Dieses Mal benutzen wir ein Dictionary, wo wir die Quadrate als Werte den Zahlen als Schlüsseln zuweisen.
- Wir fangen also mit einem leeren Doctionary `squares_1` an.
- Dann benutzen wir eine `for`-Schleife und iterieren die Variable `i` über `range(11)`.
- Im Schleifenkörper weisen wir dann `squares_1[i] = i ** 2` zu, assoziieren also die Zahl `i` mit ihrem Quadrat.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9: 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1, 18: 9, 19: 1, 20: 10}
```

Beispiel

- Dieses Mal benutzen wir ein Dictionary, wo wir die Quadrate als Werte den Zahlen als Schlüsseln zuweisen.
- Wir fangen also mit einem leeren Doctionary `squares_1` an.
- Dann benutzen wir eine `for`-Schleife und iterieren die Variable `i` über `range(11)`.
- Im Schleifenkörper weisen wir dann `squares_1[i] = i ** 2` zu, assoziieren also die Zahl `i` mit ihrem Quadrat.
- Wir können diese ganze Methode in eine einzige Dictionary Comprehension zusammenkürzen.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
    ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
          ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
          ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Wir fangen also mit einem leeren Doctionary `squares_1` an.
- Dann benutzen wir eine `for`-Schleife und iterieren die Variable `i` über `range(11)`.
- Im Schleifenkörper weisen wir dann `squares_1[i] = i ** 2` zu, assoziieren also die Zahl `i` mit ihrem Quadrat.
- Wir können diese ganze Methode in eine einzige Dictionary Comprehension zusammenkürzen.
- `{i: i ** 2 for i in range(11)}` produziert das gleiche Ergebnis.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
    ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
    ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
    ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
    ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Dann benutzen wir eine `for`-Schleife und iterieren die Variable `i` über `range(11)`.
- Im Schleifenkörper weisen wir dann `squares_1[i] = i ** 2` zu, assoziieren also die Zahl `i` mit ihrem Quadrat.
- Wir können diese ganze Methode in eine einzige Dictionary Comprehension zusammenkürzen.
- `{i: i ** 2 for i in range(11)}` produziert das gleiche Ergebnis.
- Machen wir nun etwas Spannenderes.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9: 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1, 18: 9, 19: 1, 20: 10}
```

Beispiel

- Im Schleifenkörper weisen wir dann `squares_1[i] = i ** 2` zu, assoziieren also die Zahl `i` mit ihrem Quadrat.
- Wir können diese ganze Methode in eine einzige Dictionary Comprehension zusammenkürzen.
- `{i: i ** 2 for i in range(11)}` produziert das gleiche Ergebnis.
- Machen wir nun etwas Spannenderes.
- Wir wollen ein Dictionary `maxdiv` bauen, dass den größten Divisor $m < k$ für jede Zahl k aus 2..20 hält.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9: 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1, 18: 9, 19: 1, 20: 10}
```

Beispiel

- Wir können diese ganze Methode in eine einzige Dictionary Comprehension zusammenkürzen.
- `{i: i ** 2 for i in range(11)}` produziert das gleiche Ergebnis.
- Machen wir nun etwas Spannenderes.
- Wir wollen ein Dictionary `maxdiv` bauen, dass den größten Divisor $m < k$ für jede Zahl k aus 2..20 hält.
- Wir benutzen das selbe, ineffiziente Prinzip, das wir gerade benutzt haben, um die Menge von nicht-Primzahlen zu erstellen.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"      largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
    ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
          ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
          ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- `{i: i ** 2 for i in range(11)}` produziert das gleiche Ergebnis.
- Machen wir nun etwas Spannenderes.
- Wir wollen ein Dictionary `maxdiv` bauen, dass den größten Divisor $m<k$ für jede Zahl k aus 2..20 hält.
- Wir benutzen das selbe, ineffiziente Prinzip, das wir gerade benutzt haben, um die Menge von nicht-Primzahlen zu erstellen.
- Erst lassen wir die Variable `k` über `range(21)` iterieren, lassen also `k` die Werte `0`, `1`, `2`, $\dots$, `19`, `20` annehmen.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
   ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
   ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Wir wollen ein Dictionary `maxdiv` bauen, dass den größten Divisor $m < k$ für jede Zahl k aus 2..20 hält.
- Wir benutzen das selbe, ineffiziente Prinzip, das wir gerade benutzt haben, um die Menge von nicht-Primzahlen zu erstellen.
- Erst lassen wir die Variable `k` über `range(21)` iterieren, lassen also `k` die Werte `0`, `1`, `2`, ..., `19`, `20` annehmen.
- Wir lassen eine zweite Variable `m` über `range(1, k)` iterieren, wodurch `m` die Werte `1`, `2`, ..., `k - 1` annimmt.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9: 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1, 18: 9, 19: 1, 20: 10}
```

Beispiel

- Wir benutzen das selbe, ineffiziente Prinzip, das wir gerade benutzt haben, um die Menge von nicht-Primzahlen zu erstellen.
- Erst lassen wir die Variable `k` über `range(21)` iterieren, lassen also `k` die Werte `0`, `1`, `2`, ..., `19`, `20` annehmen.
- Wir lassen eine zweite Variable `m` über `range(1, k)` iterieren, wodurch `m` die Werte `1`, `2`, ..., `k - 1` annimmt.
- Wir speichern die Zuweisung `k: m` in dem Dictionary wenn die Bedingung `k % m == 0` zutrifft, also wenn `m` die Zahl `k` ohne Rest teilen kann.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
   ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
   ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Erst lassen wir die Variable `k` über `range(21)` iterieren, lassen also `k` die Werte `0`, `1`, `2`, . . . , `19`, `20` annehmen.
- Wir lassen eine zweite Variable `m` über `range(1, k)` iterieren, wodurch `m` die Werte `1`, `2`, . . . , `k - 1` annimmt.
- Wir speichern die Zuweisung `k: m` in dem Dictionary wenn die Bedingung `k % m == 0` zutrifft, also wenn `m` die Zahl `k` ohne Rest teilen kann.
- Für die meisten `k` wird diese Bedingung von mehreren `m` erfüllt.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
   ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
   ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Wir lassen eine zweite Variable `m` über `range(1, k)` iterieren, wodurch `m` die Werte `1`, `2`, . . . , `k - 1` annimmt.
- Wir speichern die Zuweisung `k: m` in dem Dictionary wenn die Bedingung `k % m == 0` zutrifft, also wenn `m` die Zahl `k` ohne Rest teilen kann.
- Für die meisten `k` wird diese Bedingung von mehreren `m` erfüllt.
- In einem Dictionary kann aber jeder Schlüssel höchstens einmal vorkommen.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
   ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
   ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Wir lassen eine zweite Variable `m` über `range(1, k)` iterieren, wodurch `m` die Werte `1`, `2`, . . . , `k - 1` annimmt.
- Wir speichern die Zuweisung `k: m` in dem Dictionary wenn die Bedingung `k % m == 0` zutrifft, also wenn `m` die Zahl `k` ohne Rest teilen kann.
- Für die meisten `k` wird diese Bedingung von mehreren `m` erfüllt.
- In einem Dictionary kann aber jeder Schlüssel höchstens einmal vorkommen.
- In der Dictionary Comprehension bleibt daher nur die *letzte* Zuweisung bestehen.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
   ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
   ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Wir speichern die Zuweisung `k: m` in dem Dictionary wenn die Bedingung `k % m == 0` zutrifft, also wenn `m` die Zahl `k` ohne Rest teilen kann.
- Für die meisten `k` wird diese Bedingung von mehreren `m` erfüllt.
- In einem Dictionary kann aber jeder Schlüssel höchstens einmal vorkommen.
- In der Dictionary Comprehension bleibt daher nur die *letzte* Zuweisung bestehen.
- Der letzte Wert `m` dass die Bedingung für ein `k` erfüllt wird daher gespeichert.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9: 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1, 18: 9, 19: 1, 20: 10}
```

Beispiel

- Für die meisten `k` wird diese Bedingung von mehreren `m` erfüllt.
- In einem Dictionary kann aber jeder Schlüssel höchstens einmal vorkommen.
- In der Dictionary Comprehension bleibt daher nur die *letzte* Zuweisung bestehen.
- Der letzte Wert `m` dass die Bedingung für ein `k` erfüllt wird daher gespeichert.
- Da die Werte von `m` in der Iteration streng monoton steigen, ist dies der größte Divisor.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
   ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
   ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- In einem Dictionary kann aber jeder Schlüssel höchstens einmal vorkommen.
- In der Dictionary Comprehension bleibt daher nur die *letzte* Zuweisung bestehen.
- Der letzte Wert `m` dass die Bedingung für ein `k` erfüllt wird daher gespeichert.
- Da die Werte von `m` in der Iteration streng monoton steigen, ist dies der größte Divisor.
- Natürlich ist diese Methode sehr ineffizient … aber es ist ein schönes Beispiel.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"      largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
    ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
      largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
          ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
          ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- In der Dictionary Comprehension bleibt daher nur die *letzte* Zuweisung bestehen.
- Der letzte Wert `m` dass die Bedingung für ein `k` erfüllt wird daher gespeichert.
- Da die Werte von `m` in der Iteration streng monoton steigen, ist dies der größte Divisor.
- Natürlich ist diese Methode sehr ineffizient … aber es ist ein schönes Beispiel.
- Anders als in unserem Primzahl-Mengen-Beispiel, wo `m` bei `2` losging, fangen wir hier mit `m` bei `1` an.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
    ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
          ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
          ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Der letzte Wert `m` dass die Bedingung für ein `k` erfüllt wird daher gespeichert.
- Da die Werte von `m` in der Iteration streng monoton steigen, ist dies der größte Divisor.
- Natürlich ist diese Methode sehr ineffizient ... aber es ist ein schönes Beispiel.
- Anders als in unserem Primzahl-Mengen-Beispiel, wo `m` bei `2` losging, fangen wir hier mit `m` bei `1` an.
- Für jede Primzahl k' ist der größte Divisor $m < k'$ dann korrekterweise 1.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"        largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
        largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
   ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
   ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Da die Werte von `m` in der Iteration streng monoton steigen, ist dies der größte Divisor.
- Natürlich ist diese Methode sehr ineffizient … aber es ist ein schönes Beispiel.
- Anders als in unserem Primzahl-Mengen-Beispiel, wo `m` bei `2` losging, fangen wir hier mit `m` bei `1` an.
- Für jede Primzahl k' ist der größte Divisor $m < k'$ dann korrekterweise 1.
- Für 1 existiert gar kein solcher divisor – weshalb 1 auch nicht als Schlüssel in `maxdiv` auftaucht.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
   ↪ 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9:
   ↪ 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1,
   ↪ 18: 9, 19: 1, 20: 10}
```

Beispiel

- Natürlich ist diese Methode sehr ineffizient … aber es ist ein schönes Beispiel.
- Anders als in unserem Primzahl-Mengen-Beispiel, wo `m` bei `2` losging, fangen wir hier mit `m` bei `1` an.
- Für jede Primzahl k' ist der größte Divisor $m < k'$ dann korrekterweise 1.
- Für 1 existiert gar kein solcher divisor – weshalb 1 auch nicht als Schlüssel in `maxdiv` auftaucht.
- Wir geben `maxdiv` aus und es beinhaltet tatsächlich die größten Divisoren für alle Zahlen $k \leq 20$.

```python
"""Simple examples for dictionary comprehension."""

squares_1: dict[int, int] = {}  # We can start with an empty dictionary.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 9.
    squares_1[i] = i * i  # And place the square numbers in the dict.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use dictionary comprehension as follows:
squares_2: dict[int, int] = {i: i ** 2 for i in range(11)}
print(f"result of comprehension: {squares_2}")  # Print the result.

# Compute the largest divisors of the numbers in 0..20.
maxdiv: dict[int, int] = {k: m for k in range(21)
                          for m in range(1, k) if k % m == 0}
print(f"       largest divisors: {maxdiv}")
```

↓ `python3 simple_dict_comprehension.py` ↓

```
 result of construction: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
result of comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
       largest divisors: {2: 1, 3: 1, 4: 2, 5: 1, 6: 3, 7: 1, 8: 4, 9: 3, 10: 5, 11: 1, 12: 6, 13: 1, 14: 7, 15: 5, 16: 8, 17: 1, 18: 9, 19: 1, 20: 10}
```

Zusammenfassung

Zusammenfassung

- Nun haben wir das Konzept von List Comprehension also auf Mengen und Dictionaries ausgeweitet.

Zusammenfassung

- Nun haben wir das Konzept von List Comprehension also auf Mengen und Dictionaries ausgeweitet.
- Beide haben eine sehr einfache und elegante Syntax.

Zusammenfassung

- Nun haben wir das Konzept von List Comprehension also auf Mengen und Dictionaries ausgeweitet.
- Beide haben eine sehr einfache und elegante Syntax.
- Sie sind nützliche Konzepte, die uns helfen, besseren Kode zu schreiben.

谢谢您们！

Thank you!

Vielen Dank!

References I

[1] Richard Crandall und Carl Pomerance. *Prime Numbers: A Computational Perspective*. 2. Aufl. New York, NY, USA: Springer New York, 4. Aug. 2005. ISBN: 978-0-387-25282-7. doi:10.1007/0-387-28979-8 (siehe S. 15–34).

[2] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 88).

[3] 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. 88).

[4] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 88).

[5] Hans Riesel. *Prime Numbers and Computer Methods for Factorization*. 2. Aufl. Progress in Mathematics (PM). New York, NY, USA: Springer Science+Business Media, LLC, 1. Okt. 1994–30. Sep. 2012. ISSN: 0743-1643. ISBN: 978-0-8176-3743-9. doi:10.1007/978-1-4612-0251-6. Boston, MA, USA: Birkhäuser (siehe S. 15–34).

[6] Barry Warsaw. *Dict Comprehensions*. Python Enhancement Proposal (PEP) 274. Beaverton, OR, USA: Python Software Foundation (PSF), 25. Okt. 2001. URL: https://peps.python.org/pep-0274 (besucht am 2024-11-08) (siehe S. 52–54).

[7] 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. 88).

[8] Eric Wolfgang Weisstein. "Prime Number". In: *MathWorld – A Wolfram Web Resource*. Champaign, IL, USA: Wolfram Research, Inc., 22. Aug. 2024. URL: https://mathworld.wolfram.com/PrimeNumber.html (besucht am 2024-09-24) (siehe S. 15–34).

Glossary (in English) I

Python The Python programming language[2–4,7], i.e., what you will learn about in our book[7]. Learn more at https://python.org.

$i..j$ with $i, j \in \mathbb{Z}$ and $i \leq j$ is the set that contains all integer numbers in the inclusive range from i to j. For example, 5..9 is equivalent to $\{5, 6, 7, 8, 9\}$

$\mathbb{R}$ 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 $\mathbb{Z} \subset \mathbb{R}$.