合肥大學
HEFEI UNIVERSITY



Programming with Python
35. List Comprehension

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

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

人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2026-11-05

Programming with Python



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

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



Outline



1. Einleitung
2. List Comprehension
3. Beispiele
4. Performanz
5. Zusammenfassung



Einleitung

Einleitung



- Wir können Listen erstellen, in dem wir diese als Literale hinschreiben.

Einleitung



- Wir können Listen erstellen, in dem wir diese als Literale hinschreiben.
- Zum Beispiel erstellt `[1, 2, 3, 4, 5]` eine Liste mit den ersten fünf natürlichen Zahlen.

Einleitung



- Wir können Listen erstellen, in dem wir diese als Literale hinschreiben.
- Zum Beispiel erstellt `[1, 2, 3, 4, 5]` eine Liste mit den ersten fünf natürlichen Zahlen.
- Das ist sehr nützlich, kann aber umständlich werden wenn wir entweder viele Elemente haben oder diese irgendwie transformieren wollen.

Einleitung



- Wir können Listen erstellen, in dem wir diese als Literale hinschreiben.
- Zum Beispiel erstellt `[1, 2, 3, 4, 5]` eine Liste mit den ersten fünf natürlichen Zahlen.
- Das ist sehr nützlich, kann aber umständlich werden wenn wir entweder viele Elemente haben oder diese irgendwie transformieren wollen.
- Ein Literal für eine Liste mit den ersten hundert natürlichen Zahlen zu schreiben ist umständlich.

Einleitung



- Wir können Listen erstellen, in dem wir diese als Literale hinschreiben.
- Zum Beispiel erstellt `[1, 2, 3, 4, 5]` eine Liste mit den ersten fünf natürlichen Zahlen.
- Das ist sehr nützlich, kann aber umständlich werden wenn wir entweder viele Elemente haben oder diese irgendwie transformieren wollen.
- Ein Literal für eine Liste mit den ersten hundert natürlichen Zahlen zu schreiben ist umständlich.
- So etwas wie `[log(1), log(2), log(3), log(4), log(5)]` zu schreiben sieht auch nicht sehr gut aus.

Einleitung



- Wir können Listen erstellen, in dem wir diese als Literale hinschreiben.
- Zum Beispiel erstellt `[1, 2, 3, 4, 5]` eine Liste mit den ersten fünf natürlichen Zahlen.
- Das ist sehr nützlich, kann aber umständlich werden wenn wir entweder viele Elemente haben oder diese irgendwie transformieren wollen.
- Ein Literal für eine Liste mit den ersten hundert natürlichen Zahlen zu schreiben ist umständlich.
- So etwas wie `[log(1), log(2), log(3), log(4), log(5)]` zu schreiben sieht auch nicht sehr gut aus.
- Zum Glück bietet Python die viel bequemere Syntax für *list comprehension*[278].



List Comprehension

Syntax

```
"""List Comprehension in Python."""

# Create a list from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
[expression for item in sequence]

# Create a list 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]
```

- Diese Syntax erzeugt eine neue Liste deren Inhalt das Ergebnis einem Ausdruck `expression` auf die Elemente `item` einer `sequence` sind[278].

Syntax

```
"""List Comprehension in Python."""

# Create a list from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
[expression for item in sequence]

# Create a list 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]
```

- Diese Syntax erzeugt eine neue Liste deren Inhalt das Ergebnis einem Ausdruck `expression` auf die Elemente `item` einer `sequence` sind[278].
- Sie können sich das wie eine `for`-Schleife vorstellen, bei der jede Iteration einen Wert erzeugt, der dann in einer Liste gespeichert wird.

Syntax

```
"""List Comprehension in Python."""

# Create a list from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
[expression for item in sequence]

# Create a list 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]
```

- Diese Syntax erzeugt eine neue Liste deren Inhalt das Ergebnis einem Ausdruck `expression` auf die Elemente `item` einer `sequence` sind[278].
- Sie können sich das wie eine `for`-Schleife vorstellen, bei der jede Iteration einen Wert erzeugt, der dann in einer Liste gespeichert wird.
- Z. B. `[i for i in range(10)]` erzeugt eine Liste mit den Ganzzahlen von `0` bis `9`.

Syntax

```
"""List Comprehension in Python."""

# Create a list from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
[expression for item in sequence]

# Create a list 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]
```

- Sie können sich das wie eine `for`-Schleife vorstellen, bei der jede Iteration einen Wert erzeugt, der dann in einer Liste gespeichert wird.
- Z. B. `[i for i in range(10)]` erzeugt eine Liste mit den Ganzzahlen von `0` bis `9`.
- Die List Comprehension `[i ** 2 for i in range(10)]` erzeugt dagegen eine Liste mit den Quadraten dieser Zahlen, wobei der Quadrieren hier der Ausdruck `expression` ist.

Syntax

```
"""List Comprehension in Python."""

# Create a list from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
[expression for item in sequence]

# Create a list 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]
```

- Z. B. `[i for i in range(10)]` erzeugt eine Liste mit den Ganzzahlen von `0` bis `9`.
- Die List Comprehension `[i ** 2 for i in range(10)]` erzeugt dagegen eine Liste mit den Quadraten dieser Zahlen, wobei der Quadrieren hier der Ausdruck `expression` ist.
- Wir können optional auch die Elemente auswählen, die wir in der Liste haben wollen, in dem wir eine `if`-Klausel anfügen.

Syntax

```
"""List Comprehension in Python."""

# Create a list from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
[expression for item in sequence]

# Create a list 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]
```

- Z. B. `[i for i in range(10)]` erzeugt eine Liste mit den Ganzzahlen von `0` bis `9`.
- Die List Comprehension `[i ** 2 for i in range(10)]` erzeugt dagegen eine Liste mit den Quadraten dieser Zahlen, wobei der Quadrieren hier der Ausdruck `expression` ist.
- Wir können optional auch die Elemente auswählen, die wir in der Liste haben wollen, in dem wir eine `if`-Klausel anfügen.
- Z. B. `[i for i in range(10)if i != 3]` schließt die Zahl `3` von unserer List aus.

Syntax

```
"""List Comprehension in Python."""

# Create a list from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
[expression for item in sequence]

# Create a list 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]
```

- Wir können optional auch die Elemente auswählen, die wir in der Liste haben wollen, in dem wir eine `if`-Klausel anfügen.
- Z. B. `[i for i in range(10)if i != 3]` schließt die Zahl `3` von unserer List aus.
- Interessanter Weise kann die Sequenz über die List Comprehension iteriert selbst auch so ein Comprehension-Ausdruck sein.

Syntax

```
"""List Comprehension in Python."""

# Create a list from all the items in a sequence.
# 'expression' is usually an expression whose result depends on 'item'.
[expression for item in sequence]

# Create a list 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]
```

- Z. B. `[i for i in range(10)if i != 3]` schließt die Zahl `3` von unserer List aus.
- Interessanter Weise kann die Sequenz über die List Comprehension iteriert selbst auch so ein Comprehension-Ausdruck sein.
- Es ist OK zu schreiben `[i * j for i in range(2)for j in range(2)]`, was uns die Liste `[0, 0, 0, 1]` liefert, weil `i` und `j` beide jeweils unabhängig von einander nacheinander die Werte `0` und `1` annehmen.

Beispiele

Beispiel

- Schauen wir uns das mal an.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Schauen wir uns das mal an.
- Zuerst wollen wir eine Liste mit den Quadraten der Ganzzahlen von 0 bis 10 erstellen.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Schauen wir uns das mal an.
- Zuerst wollen wir eine Liste mit den Quadraten der Ganzzahlen von 0 bis 10 erstellen.
- Bevor wir über List Comprehension gelernt haben würden wir das ganz einfach mit einer normalen `for`-Schleife machen.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Schauen wir uns das mal an.
- Zuerst wollen wir eine Liste mit den Quadraten der Ganzzahlen von 0 bis 10 erstellen.
- Bevor wir über List Comprehension gelernt haben würden wir das ganz einfach mit einer normalen `for`-Schleife machen.
- Wir würden anfangen, in dem wir eine leere Liste `squares_1` erstellen.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Schauen wir uns das mal an.
- Zuerst wollen wir eine Liste mit den Quadraten der Ganzzahlen von 0 bis 10 erstellen.
- Bevor wir über List Comprehension gelernt haben würden wir das ganz einfach mit einer normalen `for`-Schleife machen.
- Wir würden anfangen, in dem wir eine leere Liste `squares_1` erstellen.
- In der `for`-Schleife würden wir eine Variable `i` über `range(0, 11)` iterieren lassen.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Bevor wir über List Comprehension gelernt haben würden wir das ganz einfach mit einer normalen `for`-Schleife machen.
- Wir würden anfangen, in dem wir eine leere Liste `squares_1` erstellen.
- In der `for`-Schleife würden wir eine Variable `i` über `range(0, 11)` iterieren lassen.
- Im Körper der Schleife würden wir dann jeweils `i ** 2` an die Liste `squares_1` anhängen, in dem wir `squares_1.append(i ** 2)` aufrufen.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Bevor wir über List Comprehension gelernt haben würden wir das ganz einfach mit einer normalen `for`-Schleife machen.
- Wir würden anfangen, in dem wir eine leere Liste `squares_1` erstellen.
- In der `for`-Schleife würden wir eine Variable `i` über `range(0, 11)` iterieren lassen.
- Im Körper der Schleife würden wir dann jeweils `i ** 2` an die Liste `squares_1` anhängen, in dem wir `squares_1.append(i ** 2)` aufrufen.
- Das benötigt drei Zeilen von Kode. Aber es geht.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- In der `for`-Schleife würden wir eine Variable `i` über `range(0, 11)` iterieren lassen.
- Im Körper der Schleife würden wir dann jeweils `i ** 2` an die Liste `squares_1` anhängen, in dem wir `squares_1.append(i ** 2)` aufrufen.
- Das benötigt drei Zeilen von Kode. Aber es geht.
- Stattdessen können wir aber auch einfach schreiben `[j ** 2 for j in range(11)]`, was genau das selbe Ergebnis mit nur einer Zeile Kode erreicht.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Im Körper der Schleife würden wir dann jeweils `i ** 2` an die Liste `squares_1` anhängen, in dem wir `squares_1.append(i ** 2)` aufrufen.
- Das benötigt drei Zeilen von Kode. Aber es geht.
- Stattdessen können wir aber auch einfach schreiben `[j ** 2 for j in range(11)]`, was genau das selbe Ergebnis mit nur einer Zeile Kode erreicht.
- Wir können uns auch aussuchen, welche Elemente einer Sequenz wir in unsere Liste einfügen wollen, in dem wir ein `if`-Statement in der List Comprehension verwenden.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Das benötigt drei Zeilen von Kode. Aber es geht.
- Stattdessen können wir aber auch einfach schreiben `[j ** 2 for j in range(11)]`, was genau das selbe Ergebnis mit nur einer Zeile Kode erreicht.
- Wir können uns auch aussuchen, welche Elemente einer Sequenz wir in unsere Liste einfügen wollen, in dem wir ein `if`-Statement in der List Comprehension verwenden.
- Im Beispiel probieren wir das aus, in dem wir eine Liste der geraden Zahlen aus der Range 0 bis 9 erstellen.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Stattdessen können wir aber auch einfach schreiben `[j ** 2 for j in range(11)]`, was genau das selbe Ergebnis mit nur einer Zeile Kode erreicht.
- Wir können uns auch aussuchen, welche Elemente einer Sequenz wir in unsere Liste einfügen wollen, in dem wir ein `if`-Statement in der List Comprehension verwenden.
- Im Beispiel probieren wir das aus, in dem wir eine Liste der geraden Zahlen aus der Range 0 bis 9 erstellen.
- Wir lassen eine Variable `k` über `range(10)` iterieren.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Wir können uns auch aussuchen, welche Elemente einer Sequenz wir in unsere Liste einfügen wollen, in dem wir ein `if`-Statement in der List Comprehension verwenden.
- Im Beispiel probieren wir das aus, in dem wir eine Liste der geraden Zahlen aus der Range 0 bis 9 erstellen.
- Wir lassen eine Variable `k` über `range(10)` iterieren.
- `k` nimmt also die Werte `0`, `1`, `2`, ..., `8`, und `9` an.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Wir können uns auch aussuchen, welche Elemente einer Sequenz wir in unsere Liste einfügen wollen, in dem wir ein `if`-Statement in der List Comprehension verwenden.
- Im Beispiel probieren wir das aus, in dem wir eine Liste der geraden Zahlen aus der Range 0 bis 9 erstellen.
- Wir lassen eine Variable `k` über `range(10)` iterieren.
- `k` nimmt also die Werte `0`, `1`, `2`, ..., `8`, und `9` an.
- Von diesem Werten nehmen wir nur die, für die `k % 2 == 0` zutrifft.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Im Beispiel probieren wir das aus, in dem wir eine Liste der geraden Zahlen aus der Range 0 bis 9 erstellen.
- Wir lassen eine Variable `k` über `range(10)` iterieren.
- `k` nimmt also die Werte `0`, `1`, `2`, ..., `8`, und `9` an.
- Von diesem Werten nehmen wir nur die, für die `k % 2 == 0` zutrifft.
- Wir berechnen also den Rest der Division von `k` und `2`.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Wir lassen eine Variable `k` über `range(10)` iterieren.
- `k` nimmt also die Werte `0`, `1`, `2`, ..., `8`, und `9` an.
- Von diesem Werten nehmen wir nur die, für die `k % 2 == 0` zutrifft.
- Wir berechnen also den Rest der Division von `k` und `2`.
- Wenn der 0 ist, dann ist `k` durch `2` teilbar und daher gerade.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- `k` nimmt also die Werte `0`, `1`, `2`, ..., `8`, und `9` an.
- Von diesem Werten nehmen wir nur die, für die `k % 2 == 0` zutrifft.
- Wir berechnen also den Rest der Division von `k` und `2`.
- Wenn der 0 ist, dann ist `k` durch `2` teilbar und daher gerade.
- Ja, ja, ich weiß ... wir hätten das auch ohne `if` machen können, wenn wir die Range `range(0, 10, 2)` genommen hätten ... oder wenn wir gleich `list(range(0, 10, 2))` gemacht hätten... ...es ist ja nur ein Beispiel.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Von diesem Werten nehmen wir nur die, für die `k % 2 == 0` zutrifft.
- Wir berechnen also den Rest der Division von `k` und `2`.
- Wenn der 0 ist, dann ist `k` durch `2` teilbar und daher gerade.
- Ja, ja, ich weiß ... wir hätten das auch ohne `if` machen können, wenn wir die Range `range(0, 10, 2)` genommen hätten ... oder wenn wir gleich `list(range(0, 10, 2))` gemacht hätten... ...es ist ja nur ein Beispiel.
- So oder so, wir bekommen die Liste `[0, 2, 4, 6, 8]`.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Wir berechnen also den Rest der Division von `k` und `2`.
- Wenn der 0 ist, dann ist `k` durch `2` teilbar und daher gerade.
- Ja, ja, ich weiß ... wir hätten das auch ohne `if` machen können, wenn wir die Range `range(0, 10, 2)` genommen hätten ... oder wenn wir gleich `list(range(0, 10, 2))` gemacht hätten... ...es ist ja nur ein Beispiel.
- So oder so, wir bekommen die Liste `[0, 2, 4, 6, 8]`.
- Nun spielen wir noch mit verschachtelter Comprehension.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Ja, ja, ich weiß ... wir hätten das auch ohne `if` machen können, wenn wir die Range `range(0, 10, 2)` genommen hätten ... oder wenn wir gleich `list(range(0, 10, 2))` gemacht hätten... ...es ist ja nur ein Beispiel.
- So oder so, wir bekommen die Liste `[0, 2, 4, 6, 8]`.
- Nun spielen wir noch mit verschachtelter Comprehension.
- Sagen wir, dass wir zwei `Iterables` habe und alle möglichen Kombinationen ihres Output produzieren wollen.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- So oder so, wir bekommen die Liste `[0, 2, 4, 6, 8]`.
- Nun spielen wir noch mit verschachtelter Comprehension.
- Sagen wir, dass wir zwei `Iterables` habe und alle möglichen Kombinationen ihres Output produzieren wollen.
- Nebenbei: Strings sind auch `Iterables` ... wir können über ihre Zeichen iterieren.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- So oder so, wir bekommen die Liste `[0, 2, 4, 6, 8]`.
- Nun spielen wir noch mit verschachtelter Comprehension.
- Sagen wir, dass wir zwei `Iterables` habe und alle möglichen Kombinationen ihres Output produzieren wollen.
- Nebenbei: Strings sind auch `Iterables` ... wir können über ihre Zeichen iterieren.
- Nehmen wir an, wir haben eine erste Sequenz `"abc"` und die zweite ist `"xy"`.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Nun spielen wir noch mit verschachtelter Comprehension.
- Sagen wir, dass wir zwei `Iterables` habe und alle möglichen Kombinationen ihres Output produzieren wollen.
- Nebenbei: Strings sind auch `Iterables` ... wir können über ihre Zeichen iterieren.
- Nehmen wir an, wir haben eine erste Sequenz `"abc"` und die zweite ist `"xy"`.
- Wir können wir eine Liste mit allen möglichen Paaren bauen, die jeweils ein Zeichen aus jedem der beiden Strings beinhalten?

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Sagen wir, dass wir zwei `Iterables` habe und alle möglichen Kombinationen ihres Output produzieren wollen.
- Nebenbei: Strings sind auch `Iterables` ... wir können über ihre Zeichen iterieren.
- Nehmen wir an, wir haben eine erste Sequenz `"abc"` und die zweite ist `"xy"`.
- Wir können wir eine Liste mit allen möglichen Paaren bauen, die jeweils ein Zeichen aus jedem der beiden Strings beinhalten?
- Einfach: in dem wir zwei `for` Statements schreiben!

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Nebenbei: Strings sind auch `Iterables` ... wir können über ihre Zeichen iterieren.
- Nehmen wir an, wir haben eine erste Sequenz `"abc"` und die zweite ist `"xy"`.
- Wir können wir eine Liste mit allen möglichen Paaren bauen, die jeweils ein Zeichen aus jedem der beiden Strings beinhalten?
- Einfach: in dem wir zwei `for` Statements schreiben!
- Wir schreiben `[f"{m}{n}"for m in "abc"for n in "xy"]`.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Nehmen wir an, wir haben eine erste Sequenz `"abc"` und die zweite ist `"xy"`.
- Wir können wir eine Liste mit allen möglichen Paaren bauen, die jeweils ein Zeichen aus jedem der beiden Strings beinhalten?
- Einfach: in dem wir zwei `for` Statements schreiben!
- Wir schreiben `[f"{m}{n}"for m in "abc"for n in "xy"]`.
- Die Variable `m` nimmt als Wert alle Zeichen aus dem String `"abc"` nacheinander an.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Wir können wir eine Liste mit allen möglichen Paaren bauen, die jeweils ein Zeichen aus jedem der beiden Strings beinhalten?
- Einfach: in dem wir zwei `for` Statements schreiben!
- Wir schreiben `[f"{m}{n}"for m in "abc"for n in "xy"]`.
- Die Variable `m` nimmt als Wert alle Zeichen aus dem String `"abc"` nacheinander an.
- Für jeden Wert, den `m` annimmt, iteriert die Variable `n` über `"xy"` and und wird daher erst `"x"` und dann `"y"`.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Einfach: in dem wir zwei `for` Statements schreiben!
- Wir schreiben `[f"{m}{n}"for m in "abc"for n in "xy"]`.
- Die Variable `m` nimmt als Wert alle Zeichen aus dem String `"abc"` nacheinander an.
- Für jeden Wert, den `m` annimmt, iteriert die Variable `n` über `"xy"` and und wird daher erst `"x"` und dann `"y"`.
- Der f-String `f"{m}{n}"` wird für jede Kombination von `m` und `n` interpoliert.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Wir schreiben `[f"{m}{n}"for m in "abc"for n in "xy"]`.
- Die Variable `m` nimmt als Wert alle Zeichen aus dem String `"abc"` nacheinander an.
- Für jeden Wert, den `m` annimmt, iteriert die Variable `n` über `"xy"` and und wird daher erst `"x"` und dann `"y"`.
- Der f-String `f"{m}{n}"` wird für jede Kombination von `m` und `n` interpoliert.
- Als Ergebnis bekommen wir die Liste `["ax", "ay", "bx", "by", "cx", "cy"]`.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Die Variable `m` nimmt als Wert alle Zeichen aus dem String `"abc"` nacheinander an.
- Für jeden Wert, den `m` annimmt, iteriert die Variable `n` über `"xy"` and und wird daher erst `"x"` und dann `"y"`.
- Der f-String `f"{m}{n}"` wird für jede Kombination von `m` und `n` interpoliert.
- Als Ergebnis bekommen wir die Liste `["ax", "ay", "bx", "by", "cx", "cy"]`.
- Natürlich können wir Listen beliebiger Datentypen mit List Comprehension erstellen.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Für jeden Wert, den `m` annimmt, iteriert die Variable `n` über `"xy"` and und wird daher erst `"x"` und dann `"y"`.
- Der f-String `f"{m}{n}"` wird für jede Kombination von `m` und `n` interpoliert.
- Als Ergebnis bekommen wir die Liste `["ax", "ay", "bx", "by", "cx", "cy"]`.
- Natürlich können wir Listen beliebiger Datentypen mit List Comprehension erstellen.
- Dies beinhaltet Listen anderer Listen, Listen von Tupeln, Mengen, oder Dictionaries ... was immer wir wollen.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Der f-String `f"{m}{n}"` wird für jede Kombination von `m` und `n` interpoliert.
- Als Ergebnis bekommen wir die Liste `["ax", "ay", "bx", "by", "cx", "cy"]`.
- Natürlich können wir Listen beliebiger Datentypen mit List Comprehension erstellen.
- Dies beinhaltet Listen anderer Listen, Listen von Tupeln, Mengen, oder Dictionaries ... was immer wir wollen.
- Wir wiederholen den selben Ansatz wie oben und speichern die Buchstabenkombinationen diesmal als Tupel.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Als Ergebnis bekommen wir die Liste `["ax", "ay", "bx", "by", "cx", "cy"]`.
- Natürlich können wir Listen beliebiger Datentypen mit List Comprehension erstellen.
- Dies beinhaltet Listen anderer Listen, Listen von Tupeln, Mengen, oder Dictionaries ... was immer wir wollen.
- Wir wiederholen den selben Ansatz wie oben und speichern die Buchstabenkombinationen diesmal als Tupel.
- `[(o, p)for o in "abc"for p in "xy"]` macht genau das.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Beispiel

- Dies beinhaltet Listen anderer Listen, Listen von Tupeln, Mengen, oder Dictionaries . . . was immer wir wollen.
- Wir wiederholen den selben Ansatz wie oben und speichern die Buchstabenkombinationen diesmal als Tupel.
- `[(o, p)for o in "abc"for p in "xy"]` macht genau das.
- Dieser Ausdruck produziert `[("a", "x"), ("a", "y"), ("b", "x"), ("b", "y"), ("c", "x"), ("c", "y")]`.

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

squares_1: list[int] = []  # We can start with an empty list.
for i in range(11):  # Then we use a for-loop over the numbers 0 to 10.
    squares_1.append(i ** 2)  # And append the squares to the list.
print(f" result of construction: {squares_1}")  # Print the result.

# Or we use list comprehension as follows:
squares_2: list[int] = [j ** 2 for j in range(11)]
print(f"result of comprehension: {squares_1}")  # Print the result.

# A very simple example of how to use `if` in list comprehension.
even_numbers: list[int] = [k for k in range(10) if k % 2 == 0]
print(f"even numbers: {even_numbers}")
# Of course, that we just an example, I know, I know, we can also...
print(f"even numbers: {list(range(0, 10, 2))}")  #  ok, ok, yes...

combinations: list[str] = [f"{m}{n}" for m in "abc" for n in "xy"]
print(f"letter combinations: {combinations}")

nested: list[tuple[str, str]] = [(o, p) for o in "abc" for p in "xy"]
print(f"letter combinations as tuples: {nested}")
```

↓ python3 simple_list_comprehension.py ↓

```
 result of construction: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
result of comprehension: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
even numbers: [0, 2, 4, 6, 8]
even numbers: [0, 2, 4, 6, 8]
letter combinations: ['ax', 'ay', 'bx', 'by', 'cx', 'cy']
letter combinations as tuples: [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b
    ↪ ', 'y'), ('c', 'x'), ('c', 'y')]
```

Performanz

Was sagt Ruff dazu?

- Wenden wir Ruff auf unser Beispielprogramm `simple_list_comprehension.py` an.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 simple_list_comprehension.py
PERF401 Use a list comprehension to create a transformed list
 --> simple_list_comprehension.py:5:5
3 | squares_1: list[int] = []  # We can start with an empty list.
4 | for i in range(11):  # Then we use a for-loop over the numbers 0 to
    ↪ 10.
5 |     squares_1.append(i ** 2)  # And append the squares to the list.
  |     ^^^^^^^^^^^^^^^^^^^^^^^^
6 | print(f" result of construction: {squares_1}")  # Print the result.
help: Replace for loop with list comprehension
Found 1 error.
# ruff 0.17.3 failed with exit code 1.
```

Was sagt Ruff dazu?

- Wenden wir Ruff auf unser Beispielprogramm `simple_list_comprehension.py` an.
- Ruff sieht die Konstruktion der Liste über die `append`-Funktion in einer Schleife als *Performanz-Problem* an, was durch die Kennung `PERF` ausgedrückt wird.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 simple_list_comprehension.py
PERF401 Use a list comprehension to create a transformed list
 --> simple_list_comprehension.py:5:5
3 | squares_1: list[int] = []  # We can start with an empty list.
4 | for i in range(11):  # Then we use a for-loop over the numbers 0 to
    ↪ 10.
5 |     squares_1.append(i ** 2)  # And append the squares to the list.
  |     ^^^^^^^^^^^^^^^^^^^^^^^^
6 | print(f" result of construction: {squares_1}")  # Print the result.
help: Replace for loop with list comprehension
Found 1 error.
# ruff 0.17.3 failed with exit code 1.
```

Was sagt Ruff dazu?

- Wenden wir Ruff auf unser Beispielprogramm `simple_list_comprehension.py` an.
- Ruff sieht die Konstruktion der Liste über die `append`-Funktion in einer Schleife als *Performanz-Problem* an, was durch die Kennung `PERF` ausgedrückt wird.
- Das ist natürlich nur ein lösbares Problem, wenn wir auch eine andere Möglichkeit haben, die Liste zu erstellen.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 simple_list_comprehension.py
PERF401 Use a list comprehension to create a transformed list
 --> simple_list_comprehension.py:5:5
3 | squares_1: list[int] = []  # We can start with an empty list.
4 | for i in range(11):  # Then we use a for-loop over the numbers 0 to
    ↪ 10.
5 |     squares_1.append(i ** 2)  # And append the squares to the list.
  |     ^^^^^^^^^^^^^^^^^^^^^^^^
6 | print(f" result of construction: {squares_1}")  # Print the result.
help: Replace for loop with list comprehension
Found 1 error.
# ruff 0.17.3 failed with exit code 1.
```

Was sagt Ruff dazu?

- Wenden wir Ruff auf unser Beispielprogramm `simple_list_comprehension.py` an.
- Ruff sieht die Konstruktion der Liste über die `append`-Funktion in einer Schleife als *Performanz-Problem* an, was durch die Kennung `PERF` ausgedrückt wird.
- Das ist natürlich nur ein lösbares Problem, wenn wir auch eine andere Möglichkeit haben, die Liste zu erstellen.
- Nun, wir kennen eine andere Möglichkeit.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 simple_list_comprehension.py
PERF401 Use a list comprehension to create a transformed list
 --> simple_list_comprehension.py:5:5
3 | squares_1: list[int] = []  # We can start with an empty list.
4 | for i in range(11):  # Then we use a for-loop over the numbers 0 to
    ↪ 10.
5 |     squares_1.append(i ** 2)  # And append the squares to the list.
  |     ^^^^^^^^^^^^^^^^^^^^^^^^
6 | print(f" result of construction: {squares_1}")  # Print the result.
help: Replace for loop with list comprehension
Found 1 error.
# ruff 0.17.3 failed with exit code 1.
```

Was sagt Ruff dazu?

- Ruff sieht die Konstruktion der Liste über die `append`-Funktion in einer Schleife als *Performanz-Problem* an, was durch die Kennung `PERF` ausgedrückt wird.
- Das ist natürlich nur ein lösbares Problem, wenn wir auch eine andere Möglichkeit haben, die Liste zu erstellen.
- Nun, wir kennen eine andere Möglichkeit.
- Wir können die Liste via List Comprehension erstellen, was aber nicht immer geht.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 simple_list_comprehension.py
PERF401 Use a list comprehension to create a transformed list
 --> simple_list_comprehension.py:5:5
3 | squares_1: list[int] = []  # We can start with an empty list.
4 | for i in range(11):  # Then we use a for-loop over the numbers 0 to
    ↪ 10.
5 |     squares_1.append(i ** 2)  # And append the squares to the list.
  |     ^^^^^^^^^^^^^^^^^^^^^^^^
6 | print(f" result of construction: {squares_1}")  # Print the result.
help: Replace for loop with list comprehension
Found 1 error.
# ruff 0.17.3 failed with exit code 1.
```

Was sagt Ruff dazu?

- Das ist natürlich nur ein lösbares Problem, wenn wir auch eine andere Möglichkeit haben, die Liste zu erstellen.
- Nun, wir kennen eine andere Möglichkeit.
- Wir können die Liste via List Comprehension erstellen, was aber nicht immer geht.
- Wenn es geht, was in unserem Beispiel ja der Fall ist, dann sehen wir oft, dass List Comprehension kompakter ist.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 simple_list_comprehension.py
PERF401 Use a list comprehension to create a transformed list
 --> simple_list_comprehension.py:5:5
3 | squares_1: list[int] = []  # We can start with an empty list.
4 | for i in range(11):  # Then we use a for-loop over the numbers 0 to
    ↪ 10.
5 |     squares_1.append(i ** 2)  # And append the squares to the list.
  |     ^^^^^^^^^^^^^^^^^^^^^^^^
6 | print(f" result of construction: {squares_1}")  # Print the result.
help: Replace for loop with list comprehension
Found 1 error.
# ruff 0.17.3 failed with exit code 1.
```

Was sagt Ruff dazu?

- Nun, wir kennen eine andere Möglichkeit.
- Wir können die Liste via List Comprehension erstellen, was aber nicht immer geht.
- Wenn es geht, was in unserem Beispiel ja der Fall ist, dann sehen wir oft, dass List Comprehension kompakter ist.
- Kode, der kompakter ist, ist oft lesbarer und daher aus Sicht eines Softwareingenieurs, oftmals zu bevorzugen.

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 simple_list_comprehension.py
PERF401 Use a list comprehension to create a transformed list
 --> simple_list_comprehension.py:5:5
3 | squares_1: list[int] = []  # We can start with an empty list.
4 | for i in range(11):  # Then we use a for-loop over the numbers 0 to
    ↪ 10.
5 |     squares_1.append(i ** 2)  # And append the squares to the list.
  |     ^^^^^^^^^^^^^^^^^^^^^^^^
6 | print(f" result of construction: {squares_1}")  # Print the result.
help: Replace for loop with list comprehension
Found 1 error.
# ruff 0.17.3 failed with exit code 1.
```

Was sagt Ruff dazu?

- Nun, wir kennen eine andere Möglichkeit.
- Wir können die Liste via List Comprehension erstellen, was aber nicht immer geht.
- Wenn es geht, was in unserem Beispiel ja der Fall ist, dann sehen wir oft, dass List Comprehension kompakter ist.
- Kode, der kompakter ist, ist oft lesbarer und daher aus Sicht eines Softwareingenieurs, oftmals zu bevorzugen.
- Aber warum wäre die ursprüngliche Schleife ein Performanz-Problem, also irgendwie langsam?

```
$ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
    ↪ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
    ↪ ,NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
    ↪ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
    ↪ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
    ↪ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
    ↪ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
    ↪ T201,TRY003,UP035,W --line-length 79 simple_list_comprehension.py
PERF401 Use a list comprehension to create a transformed list
 --> simple_list_comprehension.py:5:5
3 | squares_1: list[int] = []  # We can start with an empty list.
4 | for i in range(11):  # Then we use a for-loop over the numbers 0 to
    ↪ 10.
5 |     squares_1.append(i ** 2)  # And append the squares to the list.
  |     ^^^^^^^^^^^^^^^^^^^^^^^^
6 | print(f" result of construction: {squares_1}")  # Print the result.
help: Replace for loop with list comprehension
Found 1 error.
# ruff 0.17.3 failed with exit code 1.
```

Prüfen wir das nach

- OK, prüfen wir das nach.

Prüfen wir das nach



- OK, prüfen wir das nach.
- Um diese Sache zu untersuchen, könnten wir versuchen, zwei Listen mit dem selben Inhalt zu erstellen.

Prüfen wir das nach



- OK, prüfen wir das nach.
- Um diese Sache zu untersuchen, könnten wir versuchen, zwei Listen mit dem selben Inhalt zu erstellen.
- Eine erstellen wir, in dem wir die `append`-Methode in einer Schleife aufrufen, die andere mit List Comprehension.

Prüfen wir das nach



- OK, prüfen wir das nach.
- Um diese Sache zu untersuchen, könnten wir versuchen, zwei Listen mit dem selben Inhalt zu erstellen.
- Eine erstellen wir, in dem wir die `append`-Methode in einer Schleife aufrufen, die andere mit List Comprehension.
- Also im Grunde das, was wir gerade gemacht haben.

Prüfen wir das nach



- OK, prüfen wir das nach.
- Um diese Sache zu untersuchen, könnten wir versuchen, zwei Listen mit dem selben Inhalt zu erstellen.
- Eine erstellen wir, in dem wir die `append`-Methode in einer Schleife aufrufen, die andere mit List Comprehension.
- Also im Grunde das, was wir gerade gemacht haben.
- Die Methode, die schneller ist, hat die bessere Performanz.

Prüfen wir das nach



- OK, prüfen wir das nach.
- Um diese Sache zu untersuchen, könnten wir versuchen, zwei Listen mit dem selben Inhalt zu erstellen.
- Eine erstellen wir, in dem wir die `append`-Methode in einer Schleife aufrufen, die andere mit List Comprehension.
- Also im Grunde das, was wir gerade gemacht haben.
- Die Methode, die schneller ist, hat die bessere Performanz.
- Nun, die Laufzeit, die etwas braucht, zu messen ist immer ein bischen schwierig.

Prüfen wir das nach



- OK, prüfen wir das nach.
- Um diese Sache zu untersuchen, könnten wir versuchen, zwei Listen mit dem selben Inhalt zu erstellen.
- Eine erstellen wir, in dem wir die `append`-Methode in einer Schleife aufrufen, die andere mit List Comprehension.
- Also im Grunde das, was wir gerade gemacht haben.
- Die Methode, die schneller ist, hat die bessere Performanz.
- Nun, die Laufzeit, die etwas braucht, zu messen ist immer ein bischen schwierig.
- Die Laufzeit eines Python-Programms hängt klar von der CPU ab, auf der es läuft.

Prüfen wir das nach



- OK, prüfen wir das nach.
- Um diese Sache zu untersuchen, könnten wir versuchen, zwei Listen mit dem selben Inhalt zu erstellen.
- Eine erstellen wir, in dem wir die `append`-Methode in einer Schleife aufrufen, die andere mit List Comprehension.
- Also im Grunde das, was wir gerade gemacht haben.
- Die Methode, die schneller ist, hat die bessere Performanz.
- Nun, die Laufzeit, die etwas braucht, zu messen ist immer ein bischen schwierig.
- Die Laufzeit eines Python-Programms hängt klar von der CPU ab, auf der es läuft.
- Sie wird auch vom Betriebssystem, dem zur Verfügung stehenden RAM, der Festplattengeschwindigkeit, und natürlich von anderen Prozessen, die zur selben Zeit auf dem Computer laufen, beeinflusst[284].

Prüfen wir das nach



- OK, prüfen wir das nach.
- Um diese Sache zu untersuchen, könnten wir versuchen, zwei Listen mit dem selben Inhalt zu erstellen.
- Eine erstellen wir, in dem wir die `append`-Methode in einer Schleife aufrufen, die andere mit List Comprehension.
- Also im Grunde das, was wir gerade gemacht haben.
- Die Methode, die schneller ist, hat die bessere Performanz.
- Nun, die Laufzeit, die etwas braucht, zu messen ist immer ein bischen schwierig.
- Die Laufzeit eines Python-Programms hängt klar von der CPU ab, auf der es läuft.
- Sie wird auch vom Betriebssystem, dem zur Verfügung stehenden RAM, der Festplattengeschwindigkeit, und natürlich von anderen Prozessen, die zur selben Zeit auf dem Computer laufen, beeinflusst[284].
- Natürlich hängt sie auch von der Version des Python-Iterpreters ab und das Ergebnis der Messung könnte also nach jedem Systemupdate anders sein.

Prüfen wir das nach



- Um diese Sache zu untersuchen, könnten wir versuchen, zwei Listen mit dem selben Inhalt zu erstellen.
- Eine erstellen wir, in dem wir die `append`-Methode in einer Schleife aufrufen, die andere mit List Comprehension.
- Also im Grunde das, was wir gerade gemacht haben.
- Die Methode, die schneller ist, hat die bessere Performanz.
- Nun, die Laufzeit, die etwas braucht, zu messen ist immer ein bischen schwierig.
- Die Laufzeit eines Python-Programms hängt klar von der CPU ab, auf der es läuft.
- Sie wird auch vom Betriebssystem, dem zur Verfügung stehenden RAM, der Festplattengeschwindigkeit, und natürlich von anderen Prozessen, die zur selben Zeit auf dem Computer laufen, beeinflusst[284].
- Natürlich hängt sie auch von der Version des Python-Iterpreters ab und das Ergebnis der Messung könnte also nach jedem Systemupdate anders sein.
- Was immer wir messen, wir müssen sehr vorsichtig sein ... aber wir probieren es trotzdem.

Prüfen wir das nach

- Die Laufzeit eines Python-Programms hängt klar von der CPU ab, auf der es läuft.
- Sie wird auch vom Betriebssystem, dem zur Verfügung stehenden RAM, der Festplattengeschwindigkeit, und natürlich von anderen Prozessen, die zur selben Zeit auf dem Computer laufen, beeinflusst[284].
- Natürlich hängt sie auch von der Version des Python-Iterpreters ab und das Ergebnis der Messung könnte also nach jedem Systemupdate anders sein.
- Was immer wir messen, wir müssen sehr vorsichtig sein ... aber wir probieren es trotzdem.

Nützliches Werkzeug

`timeit` ist ein Werkzeug um die Laufzeit von kleinen Kodestücken zu messen.

Prüfen wir das nach

- Die Laufzeit eines Python-Programms hängt klar von der CPU ab, auf der es läuft.
- Sie wird auch vom Betriebssystem, dem zur Verfügung stehenden RAM, der Festplattengeschwindigkeit, und natürlich von anderen Prozessen, die zur selben Zeit auf dem Computer laufen, beeinflusst[284].
- Natürlich hängt sie auch von der Version des Python-Iterpreters ab und das Ergebnis der Messung könnte also nach jedem Systemupdate anders sein.
- Was immer wir messen, wir müssen sehr vorsichtig sein ... aber wir probieren es trotzdem.

Nützliches Werkzeug

`timeit` ist ein Werkzeug um die Laufzeit von kleinen Kodestücken zu messen. Es ist direkt Teil von Python.

Prüfen wir das nach

- Die Laufzeit eines Python-Programms hängt klar von der CPU ab, auf der es läuft.
- Sie wird auch vom Betriebssystem, dem zur Verfügung stehenden RAM, der Festplattengeschwindigkeit, und natürlich von anderen Prozessen, die zur selben Zeit auf dem Computer laufen, beeinflusst[284].
- Natürlich hängt sie auch von der Version des Python-Iterpreters ab und das Ergebnis der Messung könnte also nach jedem Systemupdate anders sein.
- Was immer wir messen, wir müssen sehr vorsichtig sein ... aber wir probieren es trotzdem.

Nützliches Werkzeug

`timeit` ist ein Werkzeug um die Laufzeit von kleinen Kodestücken zu messen. Es ist direkt Teil von Python. Dieses Modul vermeidet eine Menge von Fallen, in die man beim Zeit-Messen oftmals hereintappt[216,266].

Prüfen wir das nach

- Sie wird auch vom Betriebssystem, dem zur Verfügung stehenden RAM, der Festplattengeschwindigkeit, und natürlich von anderen Prozessen, die zur selben Zeit auf dem Computer laufen, beeinflusst[284].
- Natürlich hängt sie auch von der Version des Python-Iterpreters ab und das Ergebnis der Messung könnte also nach jedem Systemupdate anders sein.
- Was immer wir messen, wir müssen sehr vorsichtig sein ... aber wir probieren es trotzdem.
- `timeit` erlaubt uns also, die Laufzeit eines Statements zu messen.

Nützliches Werkzeug

`timeit` ist ein Werkzeug um die Laufzeit von kleinen Kodestücken zu messen. Es ist direkt Teil von Python. Dieses Modul vermeidet eine Menge von Fallen, in die man beim Zeit-Messen oftmals hereintappt[216,266].

Prüfen wir das nach

- Natürlich hängt sie auch von der Version des Python-Iterpreters ab und das Ergebnis der Messung könnte also nach jedem Systemupdate anders sein.
- Was immer wir messen, wir müssen sehr vorsichtig sein . . . aber wir probieren es trotzdem.
- `timeit` erlaubt uns also, die Laufzeit eines Statements zu messen.
- Wir wollen messen, wie lange es dauert, eine Liste mit den geraden Zahlen aus 0..1 000 000 zu erstellen.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_append.py ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Natürlich hängt sie auch von der Version des Python-Iterpreters ab und das Ergebnis der Messung könnte also nach jedem Systemupdate anders sein.
- Was immer wir messen, wir müssen sehr vorsichtig sein ... aber wir probieren es trotzdem.
- `timeit` erlaubt uns also, die Laufzeit eines Statements zu messen.
- Wir wollen messen, wie lange es dauert, eine Liste mit den geraden Zahlen aus 0..1 000 000 zu erstellen.
- Wir implementieren dafür zuerst eine Funktion `create_by_append`, die die Liste mit der `append`-Methode in einer Schleife erstellt.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Was immer wir messen, wir müssen sehr vorsichtig sein ... aber wir probieren es trotzdem.
- `timeit` erlaubt uns also, die Laufzeit eines Statements zu messen.
- Wir wollen messen, wie lange es dauert, eine Liste mit den geraden Zahlen aus 0..1 000 000 zu erstellen.
- Wir implementieren dafür zuerst eine Funktion `create_by_append`, die die Liste mit der `append`-Methode in einer Schleife erstellt.
- In einer Schleife über `range(1_000_001)` fügen wir die geraden Zahlen an die Liste `numbers` an.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- `timeit` erlaubt uns also, die Laufzeit eines Statements zu messen.
- Wir wollen messen, wie lange es dauert, eine Liste mit den geraden Zahlen aus 0..1 000 000 zu erstellen.
- Wir implementieren dafür zuerst eine Funktion `create_by_append`, die die Liste mit der `append`-Methode in einer Schleife erstellt.
- In einer Schleife über `range(1_000_001)` fügen wir die geraden Zahlen an die Liste `numbers` an.
- Am Ende wird die Liste zurückgeliefert.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Wir implementieren dafür zuerst eine Funktion `create_by_append`, die die Liste mit der `append`-Methode in einer Schleife erstellt.
- In einer Schleife über `range(1_000_001)` fügen wir die geraden Zahlen an die Liste `numbers` an.
- Am Ende wird die Liste zurückgeliefert.
- Um die Laufzeit dieser Funktion zu messen, importieren wir zuerst die Funktion `repeat` aus dem Modul `timeit`.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- In einer Schleife über `range(1_000_001)` fügen wir die geraden Zahlen an die Liste `numbers` an.
- Am Ende wird die Liste zurückgeliefert.
- Um die Laufzeit dieser Funktion zu messen, importieren wir zuerst die Funktion `repeat` aus dem Modul `timeit`.
- Wir sagen `repeat`, dass es unsere Funktion `create_by_append` einmal (`number=1`) aufrufen und die gemessen Zeit zurückliefern soll.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- In einer Schleife über `range(1_000_001)` fügen wir die geraden Zahlen an die Liste `numbers` an.
- Am Ende wird die Liste zurückgeliefert.
- Um die Laufzeit dieser Funktion zu messen, importieren wir zuerst die Funktion `repeat` aus dem Modul `timeit`.
- Wir sagen `repeat`, dass es unsere Funktion `create_by_append` einmal (`number=1`) aufrufen und die gemessen Zeit zurückliefern soll.
- Es wird die Laufzeit dann in Sekunden als `float` liefern.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Um die Laufzeit dieser Funktion zu messen, importieren wir zuerst die Funktion `repeat` aus dem Modul `timeit`.
- Wir sagen `repeat`, dass es unsere Funktion `create_by_append` einmal (`number=1`) aufrufen und die gemessen Zeit zurückliefern soll.
- Es wird die Laufzeit dann in Sekunden als `float` liefern.
- Allerdings wäre eine einzige Messung nicht sehr verlässlich, sie könnte durch das Scheduling durch das Betriebssystem oder durch Garbage Collection vom Python-Interpreter verunreinigt werden[216].

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Wir sagen `repeat`, dass es unsere Funktion `create_by_append` einmal (`number=1`) aufrufen und die gemessen Zeit zurückliefern soll.
- Es wird die Laufzeit dann in Sekunden als `float` liefern.
- Allerdings wäre eine einzige Messung nicht sehr verlässlich, sie könnte durch das Scheduling durch das Betriebssystem oder durch Garbage Collection vom Python-Interpreter verunreinigt werden[216].
- Deshalb sagen wir der `repeat`-Funktion, 50 solcher Messungen durchzuführen (Argument `repeat=50`).

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Es wird die Laufzeit dann in Sekunden als `float` liefern.
- Allerdings wäre eine einzige Messung nicht sehr verlässlich, sie könnte durch das Scheduling durch das Betriebssystem oder durch Garbage Collection vom Python-Interpreter verunreinigt werden[216].
- Deshalb sagen wir der `repeat`-Funktion, 50 solcher Messungen durchzuführen (Argument `repeat=50`).
- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Es wird die Laufzeit dann in Sekunden als `float` liefern.
- Allerdings wäre eine einzige Messung nicht sehr verlässlich, sie könnte durch das Scheduling durch das Betriebssystem oder durch Garbage Collection vom Python-Interpreter verunreinigt werden[216].
- Deshalb sagen wir der `repeat`-Funktion, 50 solcher Messungen durchzuführen (Argument `repeat=50`).
- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.

Die Dokumentation von `timeit` sagt[266]:

> *Note: it's tempting to calculate mean and standard deviation from the result vector and report these. However, this is not very useful. In a typical case, the lowest value gives a lower bound for how fast your machine can run the given code snippet; higher values in the result vector are typically not caused by variability in Python's speed, but by other processes interfering with your timing accuracy. So the* `min()` *of the result is probably the only number you should be interested in. After that, you should look at the entire vector and apply common sense rather than statistics.*
>
> — [266], 2001

Prüfen wir das nach

- Es wird die Laufzeit dann in Sekunden als `float` liefern.
- Allerdings wäre eine einzige Messung nicht sehr verlässlich, sie könnte durch das Scheduling durch das Betriebssystem oder durch Garbage Collection vom Python-Interpreter verunreinigt werden[216].
- Deshalb sagen wir der `repeat`-Funktion, 50 solcher Messungen durchzuführen (Argument `repeat=50`).
- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.

Gute Praxis

Wenn wir die Laufzeit von Kode für bestimmte Inputs prüfen, dann ergibt es Sinn, mehrere Messungen zu machen und das Minimum der beobachteten Ergebnisse zu nehmen[266].

Prüfen wir das nach

- Es wird die Laufzeit dann in Sekunden als `float` liefern.
- Allerdings wäre eine einzige Messung nicht sehr verlässlich, sie könnte durch das Scheduling durch das Betriebssystem oder durch Garbage Collection vom Python-Interpreter verunreinigt werden[216].
- Deshalb sagen wir der `repeat`-Funktion, 50 solcher Messungen durchzuführen (Argument `repeat=50`).
- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.

Gute Praxis

Wenn wir die Laufzeit von Kode für bestimmte Inputs prüfen, dann ergibt es Sinn, mehrere Messungen zu machen und das Minimum der beobachteten Ergebnisse zu nehmen[266]. Der Grund ist, das viele Faktoren (CPU-Temperatur, andere Prozesse, ...) die Laufzeit negativ beeinflussen können.

Prüfen wir das nach

- Es wird die Laufzeit dann in Sekunden als `float` liefern.
- Allerdings wäre eine einzige Messung nicht sehr verlässlich, sie könnte durch das Scheduling durch das Betriebssystem oder durch Garbage Collection vom Python-Interpreter verunreinigt werden[216].
- Deshalb sagen wir der `repeat`-Funktion, 50 solcher Messungen durchzuführen (Argument `repeat=50`).
- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.

Gute Praxis

Wenn wir die Laufzeit von Kode für bestimmte Inputs prüfen, dann ergibt es Sinn, mehrere Messungen zu machen und das Minimum der beobachteten Ergebnisse zu nehmen[266]. Der Grund ist, das viele Faktoren (CPU-Temperatur, andere Prozesse, ...) die Laufzeit negativ beeinflussen können. Es gibt aber keinen Faktor, der unseren Kode schneller machen kann, als die Hardware zulässt.

Prüfen wir das nach

- Es wird die Laufzeit dann in Sekunden als `float` liefern.
- Allerdings wäre eine einzige Messung nicht sehr verlässlich, sie könnte durch das Scheduling durch das Betriebssystem oder durch Garbage Collection vom Python-Interpreter verunreinigt werden[216].
- Deshalb sagen wir der `repeat`-Funktion, 50 solcher Messungen durchzuführen (Argument `repeat=50`).
- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.

Gute Praxis

Wenn wir die Laufzeit von Kode für bestimmte Inputs prüfen, dann ergibt es Sinn, mehrere Messungen zu machen und das Minimum der beobachteten Ergebnisse zu nehmen[266]. Der Grund ist, das viele Faktoren (CPU-Temperatur, andere Prozesse, ...) die Laufzeit negativ beeinflussen können. Es gibt aber keinen Faktor, der unseren Kode schneller machen kann, als die Hardware zulässt. Also gibt uns das Minimum den akuratesten Eindruck darauf, wie schnell unser Kode auf unserem Computer laufen kann.

Prüfen wir das nach

- Es wird die Laufzeit dann in Sekunden als `float` liefern.
- Allerdings wäre eine einzige Messung nicht sehr verlässlich, sie könnte durch das Scheduling durch das Betriebssystem oder durch Garbage Collection vom Python-Interpreter verunreinigt werden[216].
- Deshalb sagen wir der `repeat`-Funktion, 50 solcher Messungen durchzuführen (Argument `repeat=50`).
- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.

Gute Praxis

Wenn wir die Laufzeit von Kode für bestimmte Inputs prüfen, dann ergibt es Sinn, mehrere Messungen zu machen und das Minimum der beobachteten Ergebnisse zu nehmen[266]. Der Grund ist, das viele Faktoren (CPU-Temperatur, andere Prozesse, ...) die Laufzeit negativ beeinflussen können. Es gibt aber keinen Faktor, der unseren Kode schneller machen kann, als die Hardware zulässt. Also gibt uns das Minimum den akuratesten Eindruck darauf, wie schnell unser Kode auf unserem Computer laufen kann. Beachte, dass Effekte wie Caching immer noch unsere Messwerte verunreinigen können.

Prüfen wir das nach

- Allerdings wäre eine einzige Messung nicht sehr verlässlich, sie könnte durch das Scheduling durch das Betriebssystem oder durch Garbage Collection vom Python-Interpreter verunreinigt werden[216].
- Deshalb sagen wir der `repeat`-Funktion, 50 solcher Messungen durchzuführen (Argument `repeat=50`).
- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.
- Also machen wir genau das.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Deshalb sagen wir der `repeat`-Funktion, 50 solcher Messungen durchzuführen (Argument `repeat=50`).
- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.
- Also machen wir genau das.
- `repeat` gibt uns eine Liste mit gemessenen Laufzeiten.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.
- Also machen wir genau das.
- `repeat` gibt uns eine Liste mit gemessenen Laufzeiten.
- Die Funktion `min` akzeptiert eine Sequenz von Elementen und gibt uns das kleinste Element daraus zurück.[40]

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Dadurch bekommen wir alle gemessenen Laufzeiten als `list[float]`.
- Also machen wir genau das.
- `repeat` gibt uns eine Liste mit gemessenen Laufzeiten.
- Die Funktion `min` akzeptiert eine Sequenz von Elementen und gibt uns das kleinste Element daraus zurück.[40]
- Also drucken wir das Ergebnis von `min` angewandt auf die von `repeat` zurückgelieferte Liste.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Also machen wir genau das.
- `repeat` gibt uns eine Liste mit gemessenen Laufzeiten.
- Die Funktion `min` akzeptiert eine Sequenz von Elementen und gibt uns das kleinste Element daraus zurück.[40]
- Also drucken wir das Ergebnis von `min` angewandt auf die von `repeat` zurückgelieferte Liste.
- Wir formatieren die Ausgabe als Millisekunden gerundet auf drei Dezimalstellen, damit es etwas lesbarer wird.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- `repeat` gibt uns eine Liste mit gemessenen Laufzeiten.
- Die Funktion `min` akzeptiert eine Sequenz von Elementen und gibt uns das kleinste Element daraus zurück.[40]
- Also drucken wir das Ergebnis von `min` angewandt auf die von `repeat` zurückgelieferte Liste.
- Wir formatieren die Ausgabe als Millisekunden gerundet auf drei Dezimalstellen, damit es etwas lesbarer wird.
- Nun wird das Portable Document Format (PDF)-Dokument dieser Slides automatisch von einer GitHub Action[52] gebaut.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Die Funktion `min` akzeptiert eine Sequenz von Elementen und gibt uns das kleinste Element daraus zurück.[40]
- Also drucken wir das Ergebnis von `min` angewandt auf die von `repeat` zurückgelieferte Liste.
- Wir formatieren die Ausgabe als Millisekunden gerundet auf drei Dezimalstellen, damit es etwas lesbarer wird.
- Nun wird das Portable Document Format (PDF)-Dokument dieser Slides automatisch von einer GitHub Action[52] gebaut.
- Diese führt alle Beispielprogramme aus und webt ihren Output in die Slides.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Also drucken wir das Ergebnis von `min` angewandt auf die von `repeat` zurückgelieferte Liste.
- Wir formatieren die Ausgabe als Millisekunden gerundet auf drei Dezimalstellen, damit es etwas lesbarer wird.
- Nun wird das Portable Document Format (PDF)-Dokument dieser Slides automatisch von einer GitHub Action[52] gebaut.
- Diese führt alle Beispielprogramme aus und webt ihren Output in die Slides.
- Dies wird wiederholt, jedesmal wenn ich die Slides update.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Wir formatieren die Ausgabe als Millisekunden gerundet auf drei Dezimalstellen, damit es etwas lesbarer wird.
- Nun wird das Portable Document Format (PDF)-Dokument dieser Slides automatisch von einer GitHub Action[52] gebaut.
- Diese führt alle Beispielprogramme aus und webt ihren Output in die Slides.
- Dies wird wiederholt, jedesmal wenn ich die Slides update.
- Das bedeutet, dass wenn ich diesen Text schreibe, ich nicht wissen kann, welchen Wert Sie letztendlich sehen werden.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_append.py ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Nun wird das Portable Document Format (PDF)-Dokument dieser Slides automatisch von einer GitHub Action[52] gebaut.
- Diese führt alle Beispielprogramme aus und webt ihren Output in die Slides.
- Dies wird wiederholt, jedesmal wenn ich die Slides update.
- Das bedeutet, dass wenn ich diesen Text schreibe, ich nicht wissen kann, welchen Wert Sie letztendlich sehen werden.
- Auf meiner lokalen Maschine habe ich `runtime/call: 30.1 ms.` bekommen.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Diese führt alle Beispielprogramme aus und webt ihren Output in die Slides.
- Dies wird wiederholt, jedesmal wenn ich die Slides update.
- Das bedeutet, dass wenn ich diesen Text schreibe, ich nicht wissen kann, welchen Wert Sie letztendlich sehen werden.
- Auf meiner lokalen Maschine habe ich `runtime/call: 30.1 ms.` bekommen.
- Egal.

```python
"""Measure the runtime of list construction via the append method."""

from timeit import repeat  # needed for measuring the runtime


def create_by_append() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    numbers: list[int] = []
    for i in range(1_000_001):
        if i % 2 == 0:
            numbers.append(i)
    return numbers


# Perform 50 repetitions of 1 execution of create_by_append.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(create_by_append, number=1, repeat=50))
print("==== iterative list construction via append ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ `python3 list_of_numbers_append.py` ↓

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Dies wird wiederholt, jedesmal wenn ich die Slides update.
- Das bedeutet, dass wenn ich diesen Text schreibe, ich nicht wissen kann, welchen Wert Sie letztendlich sehen werden.
- Auf meiner lokalen Maschine habe ich `runtime/call: 30.1 ms.` bekommen.
- Egal.
- Um nun zu testen, ob List Comprehension nun wirklich schneller als die iterative Listenkonstruktion via `append` ist, schreiben wir Programm `list_of_numbers_comprehension.py`.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Auf meiner lokalen Maschine habe ich `runtime/call: 30.1 ms.` bekommen.
- Egal.
- Um nun zu testen, ob List Comprehension nun wirklich schneller als die iterative Listenkonstruktion via `append` ist, schreiben wir Programm `list_of_numbers_comprehension.py`.
- Dieses Programm macht fast das gleiche wie Programm `list_of_numbers_append.py` von eben.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Egal.
- Um nun zu testen, ob List Comprehension nun wirklich schneller als die iterative Listenkonstruktion via `append` ist, schreiben wir Programm `list_of_numbers_comprehension.py`.
- Dieses Programm macht fast das gleiche wie Programm `list_of_numbers_append.py` von eben.
- Es definiert eine Funktion `create_by_comprehension`, die die selbe Liste wie `create_by_append` erstellt.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Egal.
- Um nun zu testen, ob List Comprehension nun wirklich schneller als die iterative Listenkonstruktion via `append` ist, schreiben wir Programm `list_of_numbers_comprehension.py`.
- Dieses Programm macht fast das gleiche wie Programm `list_of_numbers_append.py` von eben.
- Es definiert eine Funktion `create_by_comprehension`, die die selbe Liste wie `create_by_append` erstellt.
- Nur das die Funktion List Comprehension verwendet.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Dieses Programm macht fast das gleiche wie Programm `list_of_numbers_append.py` von eben.
- Es definiert eine Funktion `create_by_comprehension`, die die selbe Liste wie `create_by_append` erstellt.
- Nur das die Funktion List Comprehension verwendet.
- Wir messen die Laufzeit dieser Funktion genauso wie vorhin.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Dieses Programm macht fast das gleiche wie Programm `list_of_numbers_append.py` von eben.
- Es definiert eine Funktion `create_by_comprehension`, die die selbe Liste wie `create_by_append` erstellt.
- Nur das die Funktion List Comprehension verwendet.
- Wir messen die Laufzeit dieser Funktion genauso wie vorhin.
- Natürlich ist das Ergebnis jedesmal anders, wenn ich die Slides update.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Es definiert eine Funktion `create_by_comprehension`, die die selbe Liste wie `create_by_append` erstellt.
- Nur das die Funktion List Comprehension verwendet.
- Wir messen die Laufzeit dieser Funktion genauso wie vorhin.
- Natürlich ist das Ergebnis jedesmal anders, wenn ich die Slides update.
- Auf meiner lokalen Machine habe ich `runtime/call: 28.7 ms.` bekommen.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Nur das die Funktion List Comprehension verwendet.
- Wir messen die Laufzeit dieser Funktion genauso wie vorhin.
- Natürlich ist das Ergebnis jedesmal anders, wenn ich die Slides update.
- Auf meiner lokalen Machine habe ich `runtime/call: 28.7 ms.` bekommen.
- Das bestätigt, das List Comprehension tatsächlich etwas schneller als iterative Konstruktion ist.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Wir messen die Laufzeit dieser Funktion genauso wie vorhin.
- Natürlich ist das Ergebnis jedesmal anders, wenn ich die Slides update.
- Auf meiner lokalen Machine habe ich `runtime/call: 28.7 ms.` bekommen.
- Das bestätigt, das List Comprehension tatsächlich etwas schneller als iterative Konstruktion ist.
- In älteren Python-Versionen ist der Unterschied größer.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Natürlich ist das Ergebnis jedesmal anders, wenn ich die Slides update.
- Auf meiner lokalen Machine habe ich `runtime/call: 28.7 ms.` bekommen.
- Das bestätigt, das List Comprehension tatsächlich etwas schneller als iterative Konstruktion ist.
- In älteren Python-Versionen ist der Unterschied größer.
- Aber er ist auch heute noch da und messbar.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Prüfen wir das nach

- Auf meiner lokalen Machine habe ich `runtime/call: 28.7 ms.` bekommen.
- Das bestätigt, das List Comprehension tatsächlich etwas schneller als iterative Konstruktion ist.
- In älteren Python-Versionen ist der Unterschied größer.
- Aber er ist auch heute noch da und messbar.
- Aber selbst wenn es keinen Unterschied gäbe ... List Comprehension ist kürzerer und eleganterer Kode.

```python
"""Measure the runtime of list construction via list comprehension."""

from timeit import repeat  # needed for measuring the runtime


def create_by_comprehension() -> list[int]:
    """
    Create the list of even numbers within 0..1'000'000.

    :return: the list of even numbers within 0..1'000'000
    """
    return [i for i in range(1_000_001) if i % 2 == 0]


# Perform 50 repetitions of 1 execution of create_by_comprehension.
# Obtain the minimum runtime of any execution as the lower bound of how
# fast this code can run.
time_in_s: float = min(repeat(
    create_by_comprehension, number=1, repeat=50))
print("==== list comprehension ====")
print(f"runtime/call: {1000 * time_in_s:.3} ms.")  # Print the result.
```

↓ python3 list_of_numbers_comprehension.py ↓

```
==== list comprehension ====
runtime/call: 67.2 ms.
```

```
==== iterative list construction via append ====
runtime/call: 66.9 ms.
```

Zusammenfassung



Zusammenfassung



- Mit List Comprehension haben wir eine elegante, compakte, und mächtige Methode zum Erstellen von Listen kennengelernt.

Zusammenfassung



- Mit List Comprehension haben wir eine elegante, compakte, und mächtige Methode zum Erstellen von Listen kennengelernt.
- Sie generalisiert die Idee von Listen-Literalen und kombiniert sie mit verschachtelbaren `for`-Schleifen.

Zusammenfassung



- Mit List Comprehension haben wir eine elegante, compakte, und mächtige Methode zum Erstellen von Listen kennengelernt.
- Sie generalisiert die Idee von Listen-Literalen und kombiniert sie mit verschachtelbaren `for`-Schleifen.
- Nicht nur das diese Methode, Listen zu konstruieren, kompakter ist als Elemente in einer Schleife an eine Liste anzufügen, sie ist auch noch schneller.

Zusammenfassung



- Mit List Comprehension haben wir eine elegante, compakte, und mächtige Methode zum Erstellen von Listen kennengelernt.
- Sie generalisiert die Idee von Listen-Literalen und kombiniert sie mit verschachtelbaren `for`-Schleifen.
- Nicht nur das diese Methode, Listen zu konstruieren, kompakter ist als Elemente in einer Schleife an eine Liste anzufügen, sie ist auch noch schneller.
- Comprehension lässt sich auch auf ein paar andere Kollektionstypen anwenden...

谢谢您们！

Thank you!

Vielen Dank!

References I

[1] Joseph Adams, Egon Balas und Daniel Zawack. "The Shifting Bottleneck Procedure for Job Shop Scheduling". *Management Science* 34(3):391–401, März 1988. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0025-1909. doi:10.1287/mnsc.34.3.391 (siehe S. 168).

[2] Miguel F. Anjos. *The QAPLIB Challenge*. Edinburgh, Scotland, UK: Miguel F. Anjos @ University of Edinburgh, 19. Nov. 2023. URL: https://www.miguelanjos.com/projects/2999 (besucht am 2026-09-28). The document lists the 28 instances of QAPLIB[41,43] that had not yet been solved to optimality by the time of its writing. (Siehe S. 173).

[3] Denis Antipov und Benjamin Doerr. "Precise Runtime Analysis for Plateaus". In: *15th International Conference on Parallel Problem Solving from Nature (PPSN XV)*. Bd. 2. 8.–12. Sep. 2018, Coimbra, Portugal. Hrsg. von Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete und L. Darrell Whitley. Bd. 11102 der Reihe Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer, 2018, S. 117–128. ISSN: 0302-9743. ISBN: 978-3-319-99258-7. doi:10.1007/978-3-319-99259-4_10 (siehe S. 173).

[4] David Lee Applegate, Robert E. Bixby, Vašek Chvátal und William John Cook. *The Traveling Salesman Problem: A Computational Study*. 2. Aufl. Bd. 17 der Reihe Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press, 2007. ISBN: 978-0-691-12993-8. This book also discusses the Concorde Traveling Salesperson Problem (TSP) solver. (Siehe S. 165, 176).

[5] David Lee Applegate und William John Cook. "A Computational Study of the Job-Shop Scheduling Problem". *ORSA Journal on Computing* 3(2):149–156, Mai 1991. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0899-1499. doi:10.1287/ijoc.3.2.149. The Job Shop Scheduling Problem (JSSP) instances used were generated in Bonn in 1986. (Siehe S. 168).

[6] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume I: Introduction to SQL Queries*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-4-0. See also[7] (siehe S. 121, 170).

[7] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume II: In-Depth Querying*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-5-7. See also[6] (siehe S. 121, 170).

References II

[8] Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete und L. Darrell Whitley, Hrsg. *15th International Conference on Parallel Problem Solving from Nature (PPSN XV)*. Bd. 2. 8.–12. Sep. 2018, Coimbra, Portugal. Bd. 11102 der Reihe Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer, 2018. ISSN: 0302-9743. ISBN: 978-3-319-99258-7. doi:10.1007/978-3-319-99259-4.

[9] Paul Gustav Heinrich Bachmann. *Die Analytische Zahlentheorie / Dargestellt von Paul Bachmann*. Bd. Zweiter Theil der Reihe Zahlentheorie: Versuch einer Gesamtdarstellung dieser Wissenschaft in ihren Hauptttheilen. Leipzig, Sachsen, Germany: B. G. Teubner, 1894. ISBN: 978-1-4181-6963-3. URL: http://gallica.bnf.fr/ark:/12148/bpt6k994750 (besucht am 2023-12-13) (siehe S. 181).

[10] Thomas Bäck, David B. Fogel und Zbigniew „Zbyszek" Michalewicz, Hrsg. *Handbook of Evolutionary Computation*. Bristol, England, UK: IOP Publishing Ltd und Oxford, Oxfordshire, England, UK: Oxford University Press, 1997. ISBN: 978-0-7503-0392-7 (siehe S. 166, 175).

[11] Daniel J. Barrett. *Efficient Linux at the Command Line*. Sebastopol, CA, USA: O'Reilly Media, Inc., Feb. 2022. ISBN: 978-1-0981-1340-7 (siehe S. 170, 176).

[12] Daniel Bartholomew. *Learning the MariaDB Ecosystem: Enterprise-level Features for Scalability and Availability*. New York, NY, USA: Apress Media, LLC, Okt. 2019. ISBN: 978-1-4842-5514-8 (siehe S. 170).

[13] Roberto Battiti und Giampietro Tecchiolli. "Local Search with Memory: Benchmarking RTS". *OR Spectrum: Quantitative Approaches in Management. Formerly Operations-Research-Spektrum* 17:67–86, Juni 1995. London, England, UK: Springer Nature Limited. ISSN: 0171-6468. doi:10.1007/BF01719249 (siehe S. 174).

[14] Roberto Battiti und Giampietro Tecchiolli. "Simulated Annealing and Tabu Search in the Long Run: A Comparison on QAP Tasks". *Computers & Mathematics with Applications* 28(6):1–8, Sep. 1994. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0898-1221. doi:10.1016/0898-1221(94)00147-2 (siehe S. 174, 176).

[15] Roberto Battiti und Giampietro Tecchiolli. "The Reactive Tabu Search". 6(2):126–140, Mai 1994. doi:10.1287/ijoc.6.2.126. Applications of Reactive Tabu Search (RTS) to the 0-1 Knapsack and Quadratic Assignment Problem (QAP). (Siehe S. 174, 176).

References III

[16] Roberto Battiti und Giampietro Tecchiolli. "Training Neural Nets with the Reactive Tabu Search". *IEEE Transactions on Neural Networks* 6(5):1185–1200, Sep. 1995. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1045-9227. doi:10.1109/72.410361. See also "Training Neural Nets with the Reactive Tabu Search, Technical Report UTM 421, Trento, Alto Adige/Südtirol, Italy: University of Trento, 11 1993" (siehe S. 174).

[17] James C. Bean. "Genetic Algorithms and Random Keys for Sequencing and Optimization". *ORSA Journal on Computing* 6(2):107–220, Frühling 1994. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0899-1499. doi:10.1287/ijoc.6.2.154. Applies the Random Keys encoding to scheduling and the QAP. See [18]. (Siehe S. 123, 165, 174).

[18] James C. Bean. *Genetics and Random Keys for Sequencing and Optimization*. Techn. Ber. 92-43. Ann Arbor, MI, USA: University of Michigan, Department of Industrial & Operations Engineering, 30. Juni 1992. URL: https://www.math.wm.edu/~rrkinc/homework/RdmKeyGA.pdf (besucht am 2026-10-01). See [17]. (Siehe S. 123, 165, 174).

[19] John Edward Beasley. "OR-Library: Distributing Test Problems by Electronic Mail". *The Journal of the Operational Research Society (JORS)* 41:1069–1072, Nov. 1990. London, England, UK: Taylor and Francis Ltd. and London, England, UK: Springer Nature Limited. ISSN: 0160-5682. doi:10.1057/jors.1990.166. See the website[20] (siehe S. 123, 168, 172).

[20] John Edward Beasley. *Welcome to OR-Library*. Uxbridge, Middlesex, England, UK: Brunel University of London, Juni 1990–Feb. 2018. URL: https://people.brunel.ac.uk/~mastjjb/jeb/info.html (besucht am 2026-09-30). See the article[19]. Contains instances of the JSSP, 1D- and 2D-Packing, Knapsack, periodic TSPs, amongst others. (Siehe S. 123, 168, 172).

[21] Andreas Beham, Michael Affenzeller und Erik Pitzer. "Metaheuristic Algorithms for the Quadratic Assignment Problem: Performance and Comparison". In: *Innovative Technologies in Management and Science*. Hrsg. von Ryszard Klempous und Jan Nikodem. Bd. 10 der Reihe Topics in Intelligent Engineering and Informatics (TIEI). Cham, Switzerland: Springer, 2015, S. 171–190. ISSN: 2193-9411. ISBN: 978-3-319-12651-7. doi:10.1007/978-3-319-12652-4_10. Algorithms like stochastic local search (SLS)/iterated local search (ILS), Simulated Annealing (SA), Tabu Search (TS), Variable Neighborhood Search (VNS), evolutionary algorithms (EAs), Memetic Algorithms (MAs) are compared and, with the exception of SA and EA, all perform best on some instance groups of QAPLIB. (Siehe S. 173).

References IV

[22] Richard Ernest Bellman. *Dynamic Programming*. Princeton Landmarks in Mathematics and Physics. Princeton, NJ, USA: Princeton University Press, 1957–2003. ISBN: 978-0-691-14668-3 (siehe S. 181).

[23] Bengt-Erik Bengtsson. "Packing Rectangular Pieces— A Heuristic Approach". *The Computer Journal* 25(3):353–357, Aug. 1982. Oxford, Oxfordshire, England, UK: Oxford University Press. ISSN: 0010-4620. doi:10.1093/comjnl/25.3.353 (siehe S. 163).

[24] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) und Newsgroups: alt.hypertext, 6. Aug. 1991. URL: https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt (besucht am 2025-02-05) (siehe S. 179).

[25] Tim Berners-Lee, Larry Masinter und Mark P. McCahill. *Uniform Resource Locators (URL)*. Request for Comments (RFC) 1738. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Dez. 1994. URL: https://www.ietf.org/rfc/rfc1738.txt (besucht am 2025-02-05) (siehe S. 178).

[26] Alex Berson. *Client/Server Architecture*. 2. Aufl. Computer Communications Series. New York, NY, USA: McGraw-Hill, 29. März 1996. ISBN: 978-0-07-005664-0 (siehe S. 165).

[27] Muhammed Beyaz, Tansel Dökeroğlu und Ahmet Cosar. "Robust Hyper-Heuristic Algorithms for the Offline Oriented/Non-Oriented 2D Bin Packing Problems". *Applied Soft Computing (ASOC)* 36:236–245, Nov. 2015. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 1568-4946. doi:10.1016/j.asoc.2015.06.063 (siehe S. 163).

[28] Christian Bierwirth. "A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms". *OR Spectrum: Quantitative Approaches in Management. Formerly Operations-Research-Spektrum* 17(2–3):87–92, Juni 1995. London, England, UK: Springer Nature Limited. ISSN: 0171-6468. doi:10.1007/BF01719250. URL: https://www.researchgate.net/publication/240263036 (besucht am 2026-09-30). Proposes a permutations-with-repetitions based encoding for the JSSP, see also[29]. (Siehe S. 125, 136, 168, 180).

References V

[29] Christian Bierwirth, Dirk C. Mattfeld und Herbert Kopfer. "On Permutation Representations for Scheduling Problems". In: *4th International Conference on Parallel Problem Solving from Nature (PPSN IV)*. 22.–24. Sep. 1996, Berlin, Germany. Hrsg. von Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg und Hans-Paul Schwefel. Bd. 1141 der Reihe Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 1996, S. 310–318. ISSN: 0302-9743. ISBN: 978-3-540-61723-5. doi:10.1007/3-540-61723-X_995. URL: https://www.researchgate.net/publication/2753293 (besucht am 2026-09-30). Proposes a permutations-with-repetitions based encoding for the JSSP, see also[28]. (Siehe S. 124, 136, 168, 180).

[30] Jacek Błażewicz, Wolfgang Domschke und Erwin Pesch. "The Job Shop Scheduling Problem: Conventional and New Solution Techniques". *European Journal of Operational Research* 93(1):1–33, Aug. 1996. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/0377-2217(95)00362-2 (siehe S. 168).

[31] Joshua Bloch. *Effective Java*. Reading, MA, USA: Addison-Wesley Professional, Mai 2008. ISBN: 978-0-321-35668-0 (siehe S. 168).

[32] Christian Blum und Verena Schmid. "Solving the 2D Bin Packing Problem by Means of a Hybrid Evolutionary Algorithm". In: *International Conference on Computational Science (ICCS'2013)*. 5.–7. Juni 2013, Barcelona, Spain. Hrsg. von Vassil Alexandrov, Michael Lees, Valeria V. Krzhizhanovskaya, Jack J. Dongarra und Peter M. A. Sloot. Bd. 18 der Reihe Procedia Computer Science. Amsterdam, The Netherlands: Elsevier B.V., 2013, S. 899–908. ISSN: 1877-0509. doi:10.1016/j.procs.2013.05.255. An EA for the oriented two-dimensional Bin Packing Problem with free cuts (2BP|O|F) using the heuristic by Wong und Lee [296]. (Siehe S. 163).

[33] Borko Bošković und Janez Brest. "Computational Search of Long Skew-Symmetric Binary Sequences with High Merit Factors". *MENDEL – Soft Computing Journal* 28(2):17–24, Dez. 2022. Brno, Czech Republic: Brno University of Technology (Vysoké učení technické v Brně). ISSN: 1803-3814. doi:10.13164/mendel.2022.2.017 (siehe S. 169).

[34] Borko Bošković, Franc Brglez und Janez Brest. "Low-Autocorrelation Binary Sequences: On Improved Merit Factors and Runtime Predictions to Achieve Them". *Applied Soft Computing (ASOC)* 56:262–285, Juli 2017. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 1568-4946. doi:10.1016/J.ASOC.2017.02.024 (siehe S. 169).

[35] Silvia Botros und Jeremy Tinley. *High Performance MySQL*. 4. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Nov. 2021. ISBN: 978-1-4920-8051-0 (siehe S. 171).

[36] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 171).

References VI

[37] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 164).

[38] Janez Brest und Borko Bošković. "A Heuristic Algorithm for a Low Autocorrelation Binary Sequence Problem with Odd Length and High Merit Factor". *IEEE Access* 6:4127–4134, Jan. 2018. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 2169-3536. doi:10.1109/ACCESS.2018.2789916 (siehe S. 169).

[39] Florian Bruhin. *Python f-Strings*. Winterthur, Switzerland: Bruhin Software, 31. Mai 2023. URL: https://fstring.help (besucht am 2024-07-25) (siehe S. 166).

[40] "Built-in Functions". In: *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library/functions.html (besucht am 2024-12-09) (siehe S. 73–99).

[41] Rainer E. Burkard, Eranda Çela, Stefan E. Karisch, Franz Rendl, Miguel F. Anjos und Peter M. Hahn. *QAPLIB – A Quadratic Assignment Problem Library – Problem Instances and Solutions*. Edinburgh, Scotland, UK: University of Edinburgh, 20. Apr. 2012–31. März 2022. doi:10.7488/ds/3428. This is the 2012 status of the QAPLIB dataset from[43] (siehe S. 121, 126, 173).

[42] Rainer E. Burkard, Eranda Çela, Panos Miltiades Pardalos und Leonidas S. Pitsoulis. "The Quadratic Assignment Problem". In: *Handbook of Combinatorial Optimization*. Hrsg. von Panos Miltiades Pardalos, Ding-Zhu Du und Ronald Lewis Graham. 1. Aufl. Boston, MA, USA: Springer, 1998, S. 1713–1809. ISBN: 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_27 (siehe S. 173).

[43] Rainer E. Burkard, Stefan E. Karisch und Franz Rendl. "QAPLIB – A Quadratic Assignment Problem Library". *Journal of Global Optimization* 10(4):391–403, Juni 1997. London, England, UK: Springer Nature Limited. ISSN: 0925-5001. doi:10.1023/A:1008293323270. This article introduces QAPLIB, a set of benchmark problems for the QAP, now hosted at [41]. (Siehe S. 121, 126, 173).

[44] Rainer E. Burkard und Josef Offermann. "Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme". *Zeitschrift für Operations Research (ZOR). now Mathematical Methods of Operations Research (MMOR)* 21:B121–B132, Aug. 1977. Heidelberg, Baden-Württemberg, Germany: Physica-Verlag; now London, England, UK: Springer Nature Limited. ISSN: 0340-9422. doi:10.1007/BF01918175. Keyboard layout modeled as QAPs. (Siehe S. 173).

[45] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 169, 175).

References VII

[46] Jacques Carlier und Éric Pinson. "An Algorithm for Solving the Job-Shop Problem". *Management Science* 35(2):164–176, 1989. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0025-1909. doi:10.1287/mnsc.35.2.164. A branch and bound (B&B) method for the JSSP. (Siehe S. 164).

[47] Eduardo Carvalho Pinto und Carola Doerr. "A Simple Proof for the Usefulness of Crossover in Black-Box Optimization". In: *15th International Conference on Parallel Problem Solving from Nature (PPSN XV)*. Bd. 2. 8.–12. Sep. 2018, Coimbra, Portugal. Hrsg. von Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete und L. Darrell Whitley. Bd. 11102 der Reihe Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer, 2018, S. 29–41. ISSN: 0302-9743. ISBN: 978-3-319-99258-7. doi:10.1007/978-3-319-99259-4_3 (siehe S. 167).

[48] Eduardo Carvalho Pinto und Carola Doerr. *Towards a More Practice-Aware Runtime Analysis of Evolutionary Algorithms*. arXiv.org: Computing Research Repository (CoRR) abs/1812.00493. Ithaca, NY, USA: Cornell Universiy Library, 3. Dez. 2018. doi:10.48550/arXiv.1812.00493. URL: https://arxiv.org/abs/1812.00493 (besucht am 2025-08-08). arXiv:1812.00493v1 [cs.NE] 3 Dec 2018 (siehe S. 162, 167, 175).

[49] Vladimír Černý. "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm". *Journal of Optimization Theory and Applications* 45(1):41–51, Jan. 1985. New York, NY, USA: Springer Science+Business Media, LLC. ISSN: 0022-3239. doi:10.1007/BF00940812 (siehe S. 174).

[50] Noureddine Chabini und Rachid Beguenane. "FPGA-Based Designs of the Factorial Function". In: *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'2022)*. 18.–20. Sep. 2022, Halifax, NS, Canada. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2022, S. 16–20. ISBN: 978-1-6654-8432-9. doi:10.1109/CCECE49351.2022.9918302 (siehe S. 179).

[51] Donald D. Chamberlin. "50 Years of Queries". *Communications of the ACM (CACM)* 67(8):110–121, Aug. 2024. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/3649887. URL: https://cacm.acm.org/research/50-years-of-queries (besucht am 2025-01-09) (siehe S. 175).

[52] Eric Chapman. *GitHub Actions*. Birmingham, England, UK: Packt Publishing Ltd, März 2024. ISBN: 978-1-80512-862-5 (siehe S. 73–102).

References VIII

[53] Bo Chen, Chris N. Potts und Gerhard J. Woeginger. "A Review of Machine Scheduling: Complexity, Algorithms and Approximability". In: *Handbook of Combinatorial Optimization*. Hrsg. von Panos Miltiades Pardalos, Ding-Zhu Du und Ronald Lewis Graham. 1. Aufl. Boston, MA, USA: Springer, 1998, S. 1493–1641. ISBN: 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_25. See also pages 21–169 in volume 3/3 by Norwell, MA, USA: Kluwer Academic Publishers. Discusses, amongst others, that the JSSP is $\mathcal{NP}$-hard. (Siehe S. 168, 181).

[54] Stephen Chen, James Montgomery und Antonio Bolufé-Röhler. "Measuring the Curse of Dimensionality and its Effects on Particle Swarm Optimization and Differential Evolution". *Applied Intelligence – The International Journal of Research on Intelligent Systems for Real Life Complex Problems* 42:514–526, Apr. 2015. London, England, UK: Springer Nature Limited. ISSN: 0924-669X. doi:10.1007/s10489-014-0613-2 (siehe S. 181).

[55] Jiayang Chen (陈嘉阳), Zhize Wu (吴志泽), Sarah Louise Thomson und Thomas Weise (汤卫思). "Frequency Fitness Assignment: Optimization Without Bias for Good Solution Outperforms Randomized Local Search on the Quadratic Assignment Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. 20.–22. Nov. 2024, Porto, Portugal. Hrsg. von Francesco Marcelloni, Kurosh Madani, Niki van Stein und Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, S. 27–37. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888600003837 (siehe S. 166, 171, 173).

[56] T. C. Edwin Cheng, Bo Peng und Zhipeng Lü. "A Hybrid Evolutionary Algorithm to Solve the Job Shop Scheduling Problem". *Annals of Operations Research* 242(2):223–237, Juli 2016. London, England, UK: Springer Nature Limited. ISSN: 0254-5330. doi:10.1007/s10479-013-1332-5. A MA utlizing TS (siehe S. 170).

[57] Philippe Chrétienne, Edward G. Coffman, Jan Karel Lenstra und Zhen Liu, Hrsg. *Scheduling Theory and Its Applications*. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., Sep. 1995. ISBN: 978-0-471-94059-3 (siehe S. 168).

[58] Nestor M. Cid-Garcia und Yasmin A. Rios-Solis. "Positions and Covering: A Two-Stage Methodology to Obtain Optimal Solutions for the 2D-Bin Packing Problem". *PLoS ONE* 15(4):e0229358, Apr. 2020. San Francisco, CA, USA: Public Library of Science (PLOS). ISSN: 1932-6203. doi:10.1371/journal.pone.0229358. An exact method for the two-dimensional Bin Packing Problem with free cuts and rotation (2BP|R|F) and the 2BP|O|F. (Siehe S. 163).

References IX

[59] David Clinton und Christopher Negus. *Ubuntu Linux Bible*. 10. Aufl. Bible Series. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 10. Nov. 2020. ISBN: 978-1-119-72233-5 (siehe S. 176, 178).

[60] Edgar Frank „Ted“ Codd. “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM (CACM)* 13(6):377–387, Juni 1970. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/362384.362685. URL: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf (besucht am 2025-01-05) (siehe S. 174).

[61] Stephen Arthur Cook. “The Complexity of Theorem-Proving Procedures”. In: *Third Annual ACM Symposium on Theory of Computing (STOC'1971)*. 3.–5. Mai 1971, Shaker Heights, OH, USA. Hrsg. von Michael A. Harrison, Ranan B. Banerji und Jeffrey D. Ullman. New York, NY, USA: Association for Computing Machinery (ACM), 1971, S. 151–158. ISBN: 978-1-4503-7464-4. doi:10.1145/800157.805047 (siehe S. 171, 181).

[62] William John Cook. *Concorde TSP Solver*. Waterloo, ON, Canada: University of Waterloo, 1997. URL: https://math.uwaterloo.ca/tsp/concorde (besucht am 2026-09-26) (siehe S. 165).

[63] William John Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton, NJ, USA: Princeton University Press, 2012–9. Nov. 2014. ISBN: 978-0-691-16352-9 (siehe S. 165, 170).

[64] William John Cook. *World TSP*. Waterloo, ON, Canada: University of Waterloo, 2013–25. Okt. 2025. URL: http://www.math.uwaterloo.ca/tsp/world (besucht am 2026-01-03) (siehe S. 170, 176).

[65] Thomas H. Cormen, Charles E. Leiserson, Ronald Linn Rivest und Clifford Stein. “Asymptotic Notation”. In: *Introduction to Algorithms*. 3. Aufl. Cambridge, MA, USA: MIT Press, 2009. Kap. 3.1, S. 43–53. ISBN: 978-0-262-03384-8. Discusses Θ, $\mathcal{O}$, Ω, o, and ω (siehe S. 181, 182).

[66] Cemre Çubukçuoğlu, Pirouz Nourian, M. Fatih Tasgetiren, I. Sevil Sariyildiz und Shervin Azadi. “Hospital Layout Design Renovation as a Quadratic Assignment Problem with Geodesic Distances”. *Journal of Building Engineering* 44:102952, Dez. 2021. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 2352-7102. doi:10.1016/j.jobe.2021.102952 (siehe S. 173).

[67] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 175).

References X

[68] Matt David und Blake Barnhill. *How to Teach People SQL*. San Francisco, CA, USA: The Data School, Chart.io, Inc., 10. Dez. 2019–10. Apr. 2023. URL: https://dataschool.com/how-to-teach-people-sql (besucht am 2025-02-27) (siehe S. 175).

[69] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 175).

[70] Kenneth Alan De Jong. *Evolutionary Computation: A Unified Approach*. Bd. 4 der Reihe Complex Adaptive Systems. Cambridge, MA, USA: MIT Press, 2006. ISBN: 978-0-262-04194-2. URL: https://www.researchgate.net/publication/220740669 (besucht am 2025-08-08) (siehe S. 166, 175).

[71] *Definition of Operations Research*. University of Western Ontario, London, ON, Canada: International Federation of Operational Research Societies (IFORS), 2020. URL: https://www.ifors.org/what-is-or (besucht am 2026-01-01) (siehe S. 172).

[72] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebW[: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 179).

[73] Maxence Delorme, Manuel Iori und Silvano Martello. "Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms". *European Journal of Operational Research* 255(1):1–20, Nov. 2016. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2016.04.030 (siehe S. 164).

[74] Ebru Demirkol, Sanjay V. Mehta und Reha Uzsoy. "Benchmarks for Shop Scheduling Problems". *European Journal of Operational Research* 109(1):137–141, Aug. 1998. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(97)00019-2 (siehe S. 168).

[75] Miroslav Dimitrov. *New Classes of Binary Sequences with High Merit Factor*. arXiv.org: Computing Research Repository (CoRR). Ithaca, NY, USA: Cornell Universiy Library, 24. Juni 2022. doi:10.48550/arXiv.2206.12070. URL: https://arxiv.org/pdf/2206.12070 (besucht am 2026-09-21). arXiv:2206.12070v1 [cs.IT] 24 Jun 2022 (siehe S. 169).

[76] Miroslav Dimitrov, Tsonka Baitcheva und Nikolay Nikolov. "Efficient Generation of Low Autocorrelation Binary Sequences". *IEEE Signal Processing Letters* 27:341–345, 2020. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1558-2361. doi:10.1109/LSP.2020.2972127 (siehe S. 169).

References XI

[77] Slobodan Dmitrović. *Modern C for Absolute Beginners: A Friendly Introduction to the C Programming Language*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0224-9 (siehe S. 165).

[78] Benjamin Doerr und Carola Doerr. "Optimal Static and Self-Adjusting Parameter Choices for the $(1 + (\lambda, \lambda))$ Genetic Algorithm". *Algorithmica* 80(5):1658–1709, Mai 2018. London, England, UK: Springer Nature Limited. ISSN: 0178-4617. doi:10.1007/s00453-017-0354-9 (siehe S. 175).

[79] Benjamin Doerr, Carola Doerr und Franziska Ebel. "From Black-Box Complexity to Designing New Genetic Algorithms". *Theoretical Computer Science* 567:87–104, Feb. 2015. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1016/J.TCS.2014.11.028 (siehe S. 172).

[80] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang und Ofer M. ShirThomas Bäck. "Benchmarking Discrete Optimization Heuristics with IOHprofiler". In: *21st Genetic and Evolutionary Computation Conference Companion (GECCO'2019)*. 13.–17. Juli 2019, Prague, Czech Republic. Hrsg. von Manuel López-Ibáñez, Anne Auger und Thomas Stützle. New York, NY, USA: Association for Computing Machinery (ACM), 2019, S. 1798–1806. ISBN: 978-1-4503-6748-6. doi:10.1145/3319619.3326810. See also[81,82] (siehe S. 131).

[81] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang und Ofer M. ShirThomas Bäck. "Benchmarking Discrete Optimization Heuristics with IOHprofiler". *Applied Soft Computing (ASOC)* 88:106027, März 2020. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 1568-4946. doi:10.1016/j.asoc.2019.106027. See also[80,82] (siehe S. 131, 162, 168, 172).

[82] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang und Ofer M. ShirThomas Bäck. "Benchmarking Discrete Optimization Heuristics with IOHprofiler". (abs/1912.09237), 31. Jan. 2021. doi:10.48550/arXiv.1912.09237. URL: https://arxiv.org/abs/1912.09237 (besucht am 2024-06-27). arXiv:1912.09237v2 [cs.NE] 31 Jan 2021, see also[80,81] (siehe S. 131).

[83] Marco Dorigo, Mauro Birattari und Thomas Stützle. "Ant Colony Optimization". *IEEE Computational Intelligence Magazine (CIM)* 1(4):28–39, Nov. 2006. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1556-603X. doi:10.1109/MCI.2006.329691. URL: https://www.researchgate.net/publication/308953674 (besucht am 2025-07-25) (siehe S. 163).

[84] Marco Dorigo, Vittorio Maniezzo und Alberto Colorni. "Ant System: Optimization by a Colony of Cooperating Agents". *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics* 26(1):29–41, Feb. 1996. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1083-4419. doi:10.1109/3477.484436. URL: https://www.researchgate.net/publication/5589170 (besucht am 2025-07-26) (siehe S. 163).

References XII

[85] Marco Dorigo und Thomas Stützle. *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press, 2004. ISBN: 978-0-262-04219-2 (siehe S. 163).

[86] Johann Dréo, Alain Pétrowski, Patrick Siarry und Éric D. Taillard. *Metaheuristics for Hard Optimization – Methods and Case Studies*. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2006. ISBN: 978-3-540-23022-9. doi:10.1007/3-540-30966-7. Discusses SA, TS, EAs, and Ant Colony Optimization (ACO). (Siehe S. 173).

[87] Stefan Droste, Thomas Jansen und Ingo Wegener. "On the Analysis of the $(1 + 1)$ Evolutionary Algorithm". *Theoretical Computer Science* 276(1-2):51–81, Apr. 2002. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1016/S0304-3975(01)00182-7 (siehe S. 162, 169, 176).

[88] Stefan Droste, Thomas Jansen und Ingo Wegener. "Upper and Lower Bounds for Randomized Search Heuristics in Black-Box Optimization". *Theoretical Computer Science* 39(4):525–544, Juli 2006. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1007/S00224-004-1177-Z (siehe S. 172).

[89] Jacques Dutka. "The Early History of the Factorial Function". *Archive for History of Exact Sciences* 43(3):225–249, Sep. 1991. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany. ISSN: 0003-9519. doi:10.1007/BF00389433. Communicated by Umberto Bottazzini (siehe S. 179).

[90] Harald Dyckhoff und Ute Finke. *Cutting and Packing in Production and Distribution: A Typology and Bibliography*. Contributions to Management Science. Heidelberg, Baden-Württemberg, Germany: Physica-Verlag, 1992. ISSN: 1431-1941. ISBN: 978-3-642-63487-1. doi:10.1007/978-3-642-58165-6 (siehe S. 164).

[91] Russell J.T. Dyer. *Learning MySQL and MariaDB*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2015. ISBN: 978-1-4493-6290-4 (siehe S. 170, 171).

[92] Kelly Easton, George L. Nemhauser und Michael A. Trick. "The Traveling Tournament Problem Description and Benchmarks". In: *7th International Conference on Principles and Practice of Constraint Programming (CP'01)*. 26. Nov.–1. Dez. 2001, Paphos, Cyprus. Hrsg. von Toby Walsh. Bd. 2239 der Reihe Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2001, S. 580–584. ISSN: 0302-9743. ISBN: 978-3-540-42863-3. doi:10.1007/3-540-45578-7_43 (siehe S. 177).

References XIII

[93] Alwalid N. Elshafei. "Hospital Layout as a Quadratic Assignment Problem". *The Journal of the Operational Research Society (JORS)* 28(1):167–179, Apr. 1977. London, England, UK: Taylor and Francis Ltd. and London, England, UK: Springer Nature Limited. ISSN: 0160-5682. doi:10.1057/jors.1977.29 (siehe S. 173).

[94] Bernhard Eschermann und Hans-Joachim Wunderlich. "Optimized Synthesis of Self-Testable Finite State Machines". In: *20th International Symposium Fault-Tolerant Computing (FTCS'1990): Digest of Papers*. 26.–28. Juni 1990, Newcastle Upon Tyne, England, UK. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 1990. ISBN: 978-0-8186-2051-5. doi:10.1109/FTCS.1990.89393 (siehe S. 173).

[95] Thomas A. Feo und Mauricio G. C. Resende. "Greedy Randomized Adaptive Search Procedures". *Journal of Global Optimization* 6(2):109–133, März 1995. London, England, UK: Springer Nature Limited. ISSN: 0925-5001. doi:10.1007/BF01096763. Introduces Greedy Randomized Adaptive Search Procedures (GRASPs) and applies them to the set covering problem, among others, and lists many problem domains. (Siehe S. 167).

[96] Luca Ferrari und Enrico Pirozzi. *Learn PostgreSQL*. 2. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Okt. 2023. ISBN: 978-1-83763-564-1 (siehe S. 173).

[97] Paola Festa und Mauricio G. C. Resende. "An Annotated Bibliography of GRASP—Part I: Algorithms". *International Transactions on Operational Research (ITOR)* 16(1):1–24, Jan. 2009. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0969-6016. doi:10.1111/j.1475-3995.2009.00663.x. See also[98] (siehe S. 133, 167).

[98] Paola Festa und Mauricio G. C. Resende. "An Annotated Bibliography of GRASP—Part II: Applications". *International Transactions on Operational Research (ITOR)* 16(2):131–172, März 2009. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0969-6016. doi:10.1111/j.1475-3995.2009.00664.x. See also[97] (siehe S. 133, 167).

[99] Simon Fischer. "A Polynomial Upper Bound for a Mutation-Based Algorithm on the Two-Dimensional Ising Model". In: *6th Genetic and Evolutionary Computation Conference (GECCO'2004), Part I*. 26.–30. Juni 2004, Seattle, WA, USA. Hrsg. von Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens und Andrew M. Tyrrell. Bd. 3102 der Reihe Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2004, S. 1100–1112. ISSN: 0302-9743. ISBN: 978-3-540-22344-3. doi:10.1007/978-3-540-24854-5_108 (siehe S. 168).

References XIV

[100] Simon Fischer und Ingo Wegener. "The One-Dimensional Ising Model: Mutation versus Recombination". *Theoretical Computer Science* 344(2-3):208–225, Nov. 2005. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1016/J.TCS.2005.04.002 (siehe S. 168).

[101] Henry Fisher und Gerald L. Thompson. "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules". In: *Industrial Scheduling*. Hrsg. von John F. Muth und Gerald L. Thompson. Englewood Cliffs, NJ, USA: Prentice-Hall, 1963. Kap. 3.2, S. 225–251 (siehe S. 168).

[102] Charles Fleurent und Jacques A. Ferland. "Genetic Hybrids for the Quadratic Assignment Problem". In: *Quadratic Assignment and Related Problems, Proceedings of a DIMACS Workshop*. 20.–21. Mai 1993, New Brunswick, NJ, USA: Rutgers University. Hrsg. von Panos Miltiades Pardalos und Henry Wolkowicz. Providence, RI, USA: American Mathematical Society (AMS), 1994, S. 173–187. ISBN: 978-0-8218-6607-8. A MA using the TS from^{256} as local search performs well. (Siehe S. 170, 173).

[103] *PDF 32000-1:2008 – Document Management – Portable Document Format – Part 1: PDF 1.7*. 1. Aufl. San Jose, CA, USA: Adobe Systems Incorporated, 1. Juli 2008. URL: https://pdf-lib.js.org/assets/with_large_page_count.pdf (besucht am 2024-12-12) (siehe S. 172).

[104] Tobias Friedrich, Pietro Simone Oliveto, Dirk Sudholt und Carsten Witt. "Analysis of Diversity-Preserving Mechanisms for Global Exploration". *Evolutionary Computation* 17(4):455–476, Winter 2009. Cambridge, MA, USA: MIT Press. ISSN: 1063-6560. doi:10.1162/evco.2009.17.4.17401 (siehe S. 177).

[105] Tobias Friedrich, Francesco Quinzan und Markus Wagner. "Escaping Large Deceptive Basins of Attraction with Heavy-Tailed Mutation Operators". In: *20th Genetic and Evolutionary Computation Conference (GECCO'2018)*. 15.–19. Juli 2018, Kyōto, Kyōto-fu, Japan. Hrsg. von Hernán E. Aguirre und Keiki Takadama. New York, NY, USA: Association for Computing Machinery (ACM), 2018, S. 293–300. ISBN: 978-1-4503-5618-3. doi:10.1145/3205455.3205515 (siehe S. 169, 177).

[106] "Formatted String Literals". In: *Python 3 Documentation. The Python Tutorial*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. Kap. 7.1.1. URL: https://docs.python.org/3/tutorial/inputoutput.html#formatted-string-literals (besucht am 2024-07-25) (siehe S. 166).

References XV

[107] José E. Gallardo, Carlos Cotta und Antonio J. Fernández. "A Memetic Algorithm for the Low Autocorrelation Binary Sequence Problem". In: *9th Genetic and Evolutionary Computation Conference (GECCO'2007)*. 7.–11. Juli 2007, London, England, UK. Hrsg. von Hod Lipson. New York, NY, USA: Association for Computing Machinery (ACM), 2007, S. 1226–1233. ISBN: 978-1-59593-697-4. doi:10.1145/1276958.1277195 (siehe S. 169).

[108] Michael R. Garey und David Stifler Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman and Company, 1979. ISBN: 978-0-7167-1045-5 (siehe S. 176).

[109] Bhavesh Gawade. "Mastering F-Strings in Python: Efficient String Handling in Python Using Smart F-Strings". In: *C O D E B*. Mumbai, Maharashtra, India: Code B Solutions Pvt Ltd, 25. Apr.–3. Juni 2025. URL: https://code-b.dev/blog/f-strings-in-python (besucht am 2025-08-04) (siehe S. 166).

[110] Mitsuo Gen, Yasuhiro Tsujimura und Erika Kubota. "Solving Job-Shop Scheduling Problems by Genetic Algorithm". In: *Humans, Information and Technology: Proceedings of the 1994 IEEE International Conference on Systems, Man and Cybernetics (SMC'1994)*. Bd. 2. 2.–5. Okt. 1994, San Antonio, TX, USA. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 1994, S. 1577–1582. ISBN: 978-0-7803-2129-8. doi:10.1109/ICSMC.1994.400072. Proposes a permutations-with-repetitions based encoding for the JSSP. (Siehe S. 136, 168, 180).

[111] Fred Glover. "Tabu Search – Part I". *ORSA Journal on Computing* 1(3):190–206, Sommer 1989. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0899-1499. doi:10.1287/ijoc.1.3.190 (siehe S. 176).

[112] Fred Glover. "Tabu Search – Part II". *ORSA Journal on Computing* 2(1):4–32, Winter 1990. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0899-1499. doi:10.1287/ijoc.2.1.4 (siehe S. 176).

[113] Fred Glover und Gary A. Kochenberger, Hrsg. *Handbook of Metaheuristics*. Bd. 57 der Reihe International Series in Operations Research & Management Science (ISOR). Cham, Switzerland: Springer, 2003. ISSN: 0884-8289. ISBN: 978-0-306-48056-0. doi:10.1007/b101874 (siehe S. 171).

[114] Fred Glover, Éric D. Taillard und Dominique de Werra. "A User's Guide to Tabu Search". *Annals of Operations Research* 41(1):3–28, März 1993. London, England, UK: Springer Nature Limited. ISSN: 0254-5330. doi:10.1007/BF02078647. Special issue on TS. (Siehe S. 176).

References XVI

[115] David Edward Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1989. ISBN: 978-0-201-15767-3 (siehe S. 166, 175).

[116] José Fernando Gonçalves und Mauricio G. C. Resende. "A Biased Random Key Genetic Algorithm for two-dimensional Bin Packing Problem (2BP) and 3D Bin Packing Problems". *International Journal of Production Economics (IJPE)* 145(2):500–510, Okt. 2013. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0925-5273. doi:10.1016/j.ijpe.2013.04.019. Applies the Biased Random Key Genetic Algorithm (BRKGA) to the 2BP|R|F and 2BP|O|F as well as 3D packing. Continuous vectors are translated to permutations and orientation data which is then the input of a placement heuristic. (Siehe S. 163, 165).

[117] José Fernando Gonçalves und Mauricio G. C. Resende. "An Extended Akers Graphical Method with a Biased Random-Key Genetic Algorithm for Job-Shop Scheduling". *International Transactions on Operational Research (ITOR)* 21(2):215–246, März 2014. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0969-6016. doi:10.1111/itor.12044. URL: http://mauricio.resende.info/doc/brkga-jss2011.pdf (besucht am 2026-09-30). The BRKGA basically is a MA using a continuous encoding of permutations that are decoded to schedules using the method from[28,29,110,237] and then refined via a TS. (Siehe S. 165, 168).

[118] José Fernando Gonçalves und Mauricio G. C. Resende. "Biased Random-Key Genetic Algorithms for Combinatorial Optimization". *Journal of Heuristics* 17:487–525, Okt. 2010. London, England, UK: Springer Nature Limited. ISSN: 1381-1231. doi:10.1007/s10732-010-9143-1 (siehe S. 165).

[119] Michael T. Goodrich. *A Gentle Introduction to NP-Completeness*. Irvine, CA, USA: University of California, Irvine, Apr. 2022. URL: https://ics.uci.edu/~goodrich/teach/cs165/notes/NPComplete.pdf (besucht am 2025-08-01) (siehe S. 181).

[120] Olaf Górski. "Why f-strings are awesome: Performance of different string concatenation methods in Python". In: *DEV Community*. Sacramento, CA, USA: DEV Community Inc., 8. Nov. 2022. URL: https://dev.to/grski/performance-of-different-string-concatenation-methods-in-python-why-f-strings-are-awesome-2e97 (besucht am 2025-08-04) (siehe S. 166).

[121] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra und Alexander Hendrik George Rinnooy Kan. "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey". Annals of Discrete Mathematics (5):287–326. Hrsg. von Peter Ladislaw Hammer, Ellis Lane Johnson und Bernhard H. Korte, 1979. ISSN: 0167-5060. doi:10.1016/S0167-5060(08)70356-X. An early review on JSSPs and FSSPs. (Siehe S. 166, 168).

References XVII

[122] Martin Grötschel, Michael Jünger und Gerhard Reinelt. "Optimal Control of Plotting and Drilling Machines: A Case Study". *Zeitschrift für Operations Research (ZOR). now Mathematical Methods of Operations Research (MMOR)* 35(1):61–84, Jan. 1991. Heidelberg, Baden-Württemberg, Germany: Physica-Verlag; now London, England, UK: Springer Nature Limited. ISSN: 0340-9422. doi:10.1007/BF01415960. This paper tackles circuit board plotting and drilling problems by modelling (parts of) them as symmetric Traveling Salesperson Problems (STSPs). (Siehe S. 176).

[123] Gregory Z. Gutin und Abraham P. Punnen, Hrsg. *The Traveling Salesman Problem and its Variations*. Bd. 12 der Reihe Combinatorial Optimization (COOP). New York, NY, USA: Springer New York, Mai 2002. ISSN: 1388-3011. doi:10.1007/b101971 (siehe S. 176).

[124] Terry Halpin und Tony Morgan. *Information Modeling and Relational Databases*. 3. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juli 2024. ISBN: 978-0-443-23791-1 (siehe S. 174).

[125] Pierre Hansen, Nenad Mladenović und José Andrés Moreno Pérez. "Variable Neighbourhood Search: Methods and Applications. Invited Survey". *4OR* 6(4):319–360, Dez. 2008. ISSN: 1619-4500. doi:10.1007/s10288-008-0089-1. URL: https://www.researchgate.net/publication/225155945 (besucht am 2026-10-02) (siehe S. 179).

[126] Jan L. Harrington. *Relational Database Design and Implementation*. 4. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Apr. 2016. ISBN: 978-0-12-849902-3 (siehe S. 174).

[127] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 170).

[128] Matthew Helmke. *Ubuntu Linux Unleashed 2021 Edition*. 14. Aufl. Reading, MA, USA: Addison-Wesley Professional, Aug. 2020. ISBN: 978-0-13-668539-5 (siehe S. 169, 178).

[129] Keld Helsgaun. "General k-opt Submoves for the Lin–Kernighan TSP Heuristic". *Mathematical Programming Computation* 1(2-3):119–163, Juli 2009. London, England, UK: Springer Nature Limited. ISSN: 1867-2949. doi:10.1007/s12532-009-0004-6. URL: https://www2.seas.gwu.edu/~simhaweb/champalg/tsp/papers/Helsgaun2.pdf (besucht am 2026-09-26) (siehe S. 170).

[130] Keld Helsgaun. "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic". 126(1):106–130, Okt. 2000. doi:10.1016/S0377-2217(99)00284-2. URL: http://akira.ruc.dk/~keld/research/LKH/LKH-2.0/DOC/LKH_REPORT.pdf (besucht am 2026-09-26) (siehe S. 170).

References XVIII

[131] André Henning. "Praktische Job-Shop Scheduling-Probleme". Diss. Jena, Thüringen, Germany: Friedrich-Schiller-Universität Jena, Aug. 2002. URL: http://www.db-thueringen.de/servlets/DocumentServlet?id=873 (besucht am 2026-09-30). Alternate url: https://nbn-resolving.org/urn:nbn:de:gbv:27-20060809-115700-4. Applies MA-like algorithms to the JSSP. (Siehe S. 168, 170).

[132] Holger H. Hoos und Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. The Morgan Kaufmann Series in Artificial Intelligence. Amsterdam, The Netherlands: Elsevier B.V., 2005. ISBN: 978-1-55860-872-6. doi:10.1016/B978-1-55860-872-6.X5016-1 (siehe S. 171, 175).

[133] Holger H. Hoos und Thomas Stützle. *SATLIB – The Satisfiability Library*. 1.4.4. Vancouver, BC, Canada: The University of British Columbia (UBC), 11. Mai 2001. URL: https://www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html (besucht am 2026-09-20). See paper[134] (siehe S. 138, 175).

[134] Holger H. Hoos und Thomas Stützle. "SATLIB: An Online Resource for Research on SAT". In: *SAT2000 – Highlights of Satisfiability Research in the Year 2000*. Hrsg. von Ian Philip Gent, Toby Walsh und Hans van Maaren. Bd. 63 der Reihe Frontiers in Artificial Intelligence and Applications. Amsterdam, The Netherlands: IOS Press BV, Jan. 2000, S. 283–292. ISSN: 0922-6389. ISBN: 978-1-58603-061-2. See the website[133] (siehe S. 138, 175).

[135] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISSN: 1863-7310. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 173).

[136] *Information Technology – Database Languages – SQL – Part 1: Framework (SQL/Framework), Part 1*. International Standard ISO/IEC 9075-1:2023(E), Sixth Edition, (ANSI X3.135). Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Juni 2023. URL: https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_(en).zip (besucht am 2025-01-08). Consists of several parts, see https://modern-sql.com/standard for information where to obtain them. (Siehe S. 175).

[137] *Information Technology – Universal Coded Character Set (UCS)*. International Standard ISO/IEC 10646:2020. Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Dez. 2020 (siehe S. 160, 178).

References XIX

[138] Manuel Iori, Vinícius Loti de Lima, Silvano Martello, Flávio K. Miyazaw und Michele Monaci. "Exact Solution Techniques for Two-Dimensional Cutting and Packing". *European Journal of Operational Research* 289(2):399–415, März 2021. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2020.06.050. See also arXiv:2004.12619v1 [math.OC] 27 Apr 2020 at doi:10.1016/j.ejor.2020.06.050. (Siehe S. 163).

[139] Christoph Jabs, Dieter Vandesande, Jeremias Berg, Johannes Fichte, George Katsirelos, Ruben Martins, André Schidler und Dominik Schreiber. *MaxSAT Evaluation: Complete Benchmark Collection*. 2026. Aufl. Geneva, Switzerland: zenodo (hosted by European Organization for Nuclear Research (CERN)), 9. Juli 2026. doi:10.5281/zenodo.21283148. All versions of this dataset can be reached directly via doi:10.5281/zenodo.21283147. See also https://github.com/maxsat-evaluations/data and https://maxsat-evaluations.github.io. (Siehe S. 171).

[140] Dean Jacobs, Jan Prins, Peter Siegel und Kenneth Wilson. "Monte Carlo Techniques in Code Optimization". In: *15th Annual Workshop on Microprogramming (MICRO 15)*. 5.–7. Okt. 1982. Hrsg. von Joseph Allen Fisher, William J. Tracz und Bill Hopkins. Palo Alto, CA, USA: Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE) und New York, NY, USA: Association for Computing Machinery (ACM), Okt. 1982, S. 143–148. doi:10.5555/800036.800944. See [141] (siehe S. 139).

[141] Dean Jacobs, Jan Prins, Peter Siegel und Kenneth Wilson. "Monte Carlo Techniques in Code Optimization". *ACM SIGMICRO Newsletter* 13(4):143–148, Dez. 1982. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 1050-916X. doi:10.1145/1014194.800944. See [140] (siehe S. 139, 174).

[142] Stefan Jakobs. "On Genetic Algorithms for the Packing of Polygons". *European Journal of Operational Research* 88(1):165–181, Jan. 1996. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/0377-2217(94)00166-9 (siehe S. 167).

[143] Wassim Jaziri, Hrsg. *Tabu Search*. London, England, UK: IntechOpen Limited, 1. Aug. 2008. ISBN: 978-3-902613-34-9. doi:10.5772/65. URL: https://www.intechopen.com/books/6114 (besucht am 2026-09-30) (siehe S. 176).

[144] David Stifler Johnson, Gregory Z. Gutin, Lyle Andrew McGeoch, Anders Yeo, Weixiong Zhang und Alexei Zverovitch. *Experimental Analysis of Heuristics for the ATSP*. Hrsg. von Gregory Z. Gutin und Abraham P. Punnen. New York, NY, USA, Mai 2002. doi:10.1007/0-306-48213-4_10. URL: https://archive.dimacs.rutgers.edu/Challenges/TSP/papers/atspchap.pdf (besucht am 2026-09-25) (siehe S. 176).

References XX

[145] David Stifler Johnson und Lyle Andrew McGeoch. *Experimental Analysis of Heuristics for the STSP*. Hrsg. von Gregory Z. Gutin und Abraham P. Punnen. New York, NY, USA, Mai 2002. doi:10.1007/0-306-48213-4_9. URL: https://archive.dimacs.rutgers.edu/Challenges/TSP/papers/stspchap.pdf (besucht am 2026-09-25) (siehe S. 176).

[146] David Stifler Johnson, Lyle Andrew McGeoch, Fred Glover und Cesar Rego. *8th DIMACS Implementation Challenge: The Traveling Salesman Problem*. 1. Juni 2000–19. Nov. 2013. URL: https://archive.dimacs.rutgers.edu/Challenges/TSP (besucht am 2026-09-25) (siehe S. 176).

[147] Selmer Martin Johnson. "Optimal Two- and Three-Stage Production Schedules with Setup Times Included". *Naval Research Logistics Quarterly* I(1):61–68, März 1954. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0894-069X. doi:10.1002/nav.3800010110. URL: https://www.rand.org/content/dam/rand/pubs/papers/2008/P402.pdf (besucht am 2023-12-06) (siehe S. 166).

[148] Stephen Curtis Johnson. *Lint, a C Program Checker*. Computing Science Technical Report 78-1273. New York, NY, USA: Bell Telephone Laboratories, Incorporated, 25. Okt. 1978. URL: https://wolfram.schneider.org/bsd/7thEdManVol2/lint/lint.pdf (besucht am 2024-08-23) (siehe S. 169).

[149] Scott Kirkpatrick, C. Daniel Gelatt, Jr. und Mario P. Vecchi. "Optimization by Simulated Annealing". *Science Magazine* 220(4598):671–680, 13. Mai 1983. Washington, D.C., USA: American Association for the Advancement of Science (AAAS). ISSN: 0036-8075. doi:10.1126/science.220.4598.671. URL: https://www.researchgate.net/publication/6026283 (besucht am 2025-08-08) (siehe S. 174).

[150] Robert Klein. *Scheduling of Resource-Constrained Projects*. Bd. 10 der Reihe Operations Research/Computer Science Interfaces Series (ORCS). New York, NY, USA: Springer New York, 2000. ISSN: 1387-666X. ISBN: 978-0-7923-8637-7. doi:10.1007/978-1-4615-4629-0 (siehe S. 166).

[151] Donald Ervin Knuth. "Big Omicron and Big Omega and Big Theta". *ACM SIGACT News* 8(2):18–24, Apr.–Juni 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0163-5700. doi:10.1145/1008328.1008329 (siehe S. 181, 182).

[152] Donald Ervin Knuth. *Fundamental Algorithms*. 3. Aufl. Bd. 1 der Reihe The Art of Computer Programming. Reading, MA, USA: Addison-Wesley Professional, 1997. ISBN: 978-0-201-89683-1 (siehe S. 181, 182).

References XXI

[153] Donald Ervin Knuth. *Sorting and Searching*. Bd. 3 der Reihe The Art of Computer Programming. Reading, MA, USA: Addison-Wesley Professional, 1998. ISBN: 978-0-201-89685-5 (siehe S. 181).

[154] Tjalling Charles Koopmans und Martin Beckmann. "Assignment Problems and the Location of Economic Activities". *Econometrica* 25(1):53–76, 1957. New Haven, CT, USA: The Econometric Society and Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0012-9682. doi:10.2307/1907742 (siehe S. 173).

[155] Jakob Krarup und Peter Mark Pruzan. "Computer-aided Layout Design". In: *Mathematical Programming in Use*. Hrsg. von Michel Louis Balinski und Claude Lemaréchal. Bd. 9 der Reihe. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 1978. Kap. 6, S. 75–94. ISSN: 0303-3929. ISBN: 978-3-642-00796-5. doi:10.1007/BFb0120827. Uses QAPs as model for designing architectural layouts. (Siehe S. 173).

[156] Mark W. Krentel. "The Complexity of Optimization Problems". *Journal of Computer and System Sciences* 36(3):490–509, Juni 1988. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0022-0000. Explores the hardness (optimization) problems such TSP, Clique, and Coloring, where computing the optimal value is $\mathcal{NP}$-complete. (Siehe S. 176).

[157] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 175).

[158] Edmund Landau. *Handbuch der Lehre von der Verteilung der Primzahlen*. Leipzig, Sachsen, Germany: B. G. Teubner, 1909. ISBN: 978-0-8218-2650-8 (siehe S. 181).

[159] Łukasz Langa. *Literature Overview for Type Hints*. Python Enhancement Proposal (PEP) 482. Beaverton, OR, USA: Python Software Foundation (PSF), 8. Jan. 2015. URL: https://peps.python.org/pep-0482 (besucht am 2024-10-09) (siehe S. 177).

[160] Eugene Leighton Lawler, Jan Karel Lenstra und Alexander Hendrik George Rinnooy Kan. "Recent Developments in Deterministic Sequencing and Scheduling: A Survey". In: *Deterministic and Stochastic Scheduling: Proceedings of the NATO Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems*. 6.–17. Juni 1981, Durham, England, UK. Hrsg. von Michael Anthony Howe Dempster, Jan Karel Lenstra und Alexander Hendrik George Rinnooy Kan. Bd. 84 der Reihe Nato Science Series C: Mathematical and Physical Science (ASIC). London, England, UK: Springer Nature Limited, Apr. 1982, S. 35–73. ISBN: 978-90-277-1397-1. doi:10.1007/978-94-009-7801-0_3. The JSSP is $\mathcal{NP}$-hard. (Siehe S. 168).

References XXII

[161] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan und David B. Shmoys. "Sequencing and Scheduling: Algorithms and Complexity". In: *Production Planning and Inventory*. Hrsg. von Stephen C. Graves, Alexander Hendrik George Rinnooy Kan und Paul H. Zipkin. Bd. IV der Reihe Handbooks of Operations Research and Management Science. Amsterdam, The Netherlands: Elsevier B.V., 1993. Kap. 9, S. 445–522. ISSN: 0927-0507. ISBN: 978-0-444-87472-6. doi:10.1016/S0927-0507(05)80189-6. URL: http://alexandria.tue.nl/repository/books/339776.pdf (besucht am 2023-12-06). Discusses, amongst others, that the JSSP is $\mathcal{NP}$-hard. (Siehe S. 168, 181).

[162] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan und David B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, England, UK: Wiley Interscience, Sep. 1985. ISSN: 0277-2698. ISBN: 978-0-471-90413-7 (siehe S. 176).

[163] Stephen R. Lawrence. "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques". Diss. Pittsburgh, PA, USA: Graduate School of Industrial Administration (GSIA), Carnegie-Mellon University, 1984 (siehe S. 168).

[164] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISSN: 1863-7310. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 173).

[165] Michael Lee, Ivan Levkivskyi und Jukka Lehtosalo. *Literal Types*. Python Enhancement Proposal (PEP) 586. Beaverton, OR, USA: Python Software Foundation (PSF), 14. März 2019. URL: https://peps.python.org/pep-0586 (besucht am 2024-12-17) (siehe S. 170).

[166] Jukka Lehtosalo, Ivan Levkivskyi, Jared Hance, Ethan Smith, Guido van Rossum, Jelle „JelleZijlstra" Zijlstra, Michael J. Sullivan, Shantanu Jain, Xuanda Yang, Jingchen Ye, Nikita Sobolev und Mypy Contributors. *Mypy – Static Typing for Python*. San Francisco, CA, USA: GitHub Inc, 2024. URL: https://github.com/python/mypy (besucht am 2024-08-17) (siehe S. 171).

[167] Xueping Li und Kaike Zhang. "Single Batch Processing Machine Scheduling with Two-Dimensional Bin Packing Constraints". *International Journal of Production Economics (IJPE)* 196:113–121, Feb. 2018. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0925-5273. doi:10.1016/j.ijpe.2017.11.015. Mentions some scheduling tasks of Additive Manufacturing jobs can be viewed as instance of the 2BP|R|F. (Siehe S. 163).

References XXIII

[168] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Jörg Lässig, Daan van den Berg, Sarah Louise Thomson und Thomas Weise (汤卫思). "Addressing the Traveling Salesperson Problem with Frequency Fitness Assignment and Hybrid Algorithms". *Soft Computing* 28(17-18):9495–9508, Juli 2024. London, England, UK: Springer Nature Limited. ISSN: 1432-7643. doi:10.1007/S00500-024-09718-8 (siehe S. 166, 171, 176).

[169] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Jörg Lässig, Daan van den Berg und Thomas Weise (汤卫思). "Solving the Traveling Salesperson Problem using Frequency Fitness Assignment". In: *IEEE Symposium Series on Computational Intelligence (SSCI'2022)*. 4.–7. Dez. 2022, Singapore. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2022. ISBN: 978-1-6654-8769-6. doi:10.1109/SSCI51031.2022.10022296 (siehe S. 166, 171).

[170] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Matthias Thürer, Markus Wagner und Thomas Weise (汤卫思). "Generating Small Instances with Interesting Features for the Traveling Salesperson Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. 20.–22. Nov. 2024, Porto, Portugal. Hrsg. von Francesco Marcelloni, Kurosh Madani, Niki van Stein und Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, S. 173–180. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888800003837 (siehe S. 171).

[171] Shen Lin und Brian W. Kernighan. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". *Operations Research* 21(2):498–516, März–Apr. 1973. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0030-364X. doi:10.1287/opre.21.2.498. URL: https://www.cs.princeton.edu/~bwk/btl.mirror/tsp.pdf (besucht am 2026-09-26) (siehe S. 170).

[172] John D. C. Little, Katta G. Murty, Dura W. Sweeny und Caroline Karel. *An Algorithm for the Traveling Salesman Problem*. Sloan Working Papers 07-63. Massachusetts Institute of Technology (MIT): Massachusetts Institute of Technology (MIT), Sloan School of Management, März 1963. URL: http://dspace.mit.edu/bitstream/handle/1721.1/46828/algorithmfortrav00litt.pdf (besucht am 2026-01-03) (siehe S. 164).

[173] Dequan Liu (刘德全) und Hongfei Teng (滕弘飞). "An Improved BL-Algorithm for EAGenetic Algorithm of the Orthogonal Packing of Rectangles". *European Journal of Operational Research* 112(2):413–420, Jan. 1999. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(97)00437-2 (siehe S. 167).

References XXIV

[174] Andrea Lodi, Silvano Martello und Michele Monaci. "Two-Dimensional Packing Problems: A Survey". *European Journal of Operational Research* 141(2):241–252, Sep. 2002. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(02)00123-6 (siehe S. 163, 164).

[175] Andrea Lodi, Silvano Martello und Daniele Vigo. "Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems". *INFORMS Journal on Computing* 11(4):345–357, Herbst 1999. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 1091-9856. doi:10.1287/ijoc.11.4.345 (siehe S. 163).

[176] Andrea Lodi, Silvano Martello und Daniele Vigo. "Recent Advances on Two-Dimensional Bin Packing Problems". *Discrete Applied Mathematics* 123(1–3):379–396, Nov. 2002. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0166-218X. doi:10.1016/S0166-218X(01)00347-X (siehe S. 163–165).

[177] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter M. Hahn und Tania Querido. "A Survey for the Quadratic Assignment Problem". *European Journal of Operational Research* 176(2):657–690, 2007. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2005.09.032 (siehe S. 173).

[178] Mariana A. Londe, Luciana S. Pessoa, Carlos E. Andrade und Mauricio G. C. Resende. *Biased Random-Key Genetic Algorithms: A Review*. arXiv.org: Computing Research Repository (CoRR) abs/2312.00961v2. Ithaca, NY, USA: Cornell Universiy Library, 6. Dez. 2023. doi:10.48550/arXiv.2312.00961. URL: https://arxiv.org/abs/2312.00961v2 (besucht am 2024-06-27). arXiv:2312.00961v2 [cs.NE] 06 Dec 2023; See [179] (siehe S. 144).

[179] Mariana A. Londe, Luciana S. Pessoa, Carlos E. Andrade und Mauricio G. C. Resende. "Biased Random-Key Genetic Algorithms: A Review". *European Journal of Operational Research* 321(1):1–22, Feb. 2025. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2024.03.030. See [178] (siehe S. 144, 165).

[180] Gloria Lotha, Aakanksha Gaur, Erik Gregersen, Swati Chopra und William L. Hosch. "Client-Server Architecture". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 3. Jan. 2025. URL: https://www.britannica.com/technology/client-server-architecture (besucht am 2025-01-20) (siehe S. 165).

[181] Marc Loy, Patrick Niemeyer und Daniel Leuck. *Learning Java*. 5. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2020. ISBN: 978-1-4920-5627-0 (siehe S. 168).

References XXV

[182] Peter Luschny. *A New Kind of Factorial Function*. Highland Park, NJ, USA: The OEIS Foundation Inc., 4. Okt. 2015. URL: https://oeis.org/A000142/a000142.pdf (besucht am 2024-09-29) (siehe S. 179).

[183] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 173).

[184] Rita Macedo, Cláudio Alves und J. M. Valério de Carvalho. "Arc-Flow Model for the Two-Dimensional Guillotine Cutting Stock Problem". *Computers & Operations Research* 37(6):991–1001, Juni 2010. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0305-0548. doi:10.1016/j.cor.2009.08.005 (siehe S. 165).

[185] Reinhard Männer und Bernard Manderick, Hrsg. *Parallel Problem Solving from Nature 2 (PPSN-II)*. 28.–30. Sep. 1992, Brussels, Belgium. Amsterdam, The Netherlands: Elsevier B.V., 1992.

[186] Francesco Marcelloni, Kurosh Madani, Niki van Stein und Joaquim Filipe, Hrsg. *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. 20.–22. Nov. 2024, Porto, Portugal. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0000195000003837.

[187] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: https://mariadb.com/kb/en/documentation (besucht am 2025-04-24) (siehe S. 170).

[188] Charlie Marsh. "Ruff". In: URL: https://pypi.org/project/ruff (besucht am 2025-08-29) (siehe S. 174).

[189] Charlie Marsh. *ruff: An Extremely Fast Python Linter and Code Formatter, Written in Rust*. New York, NY, USA: Astral Software Inc., 28. Aug. 2022. URL: https://docs.astral.sh/ruff (besucht am 2024-08-23) (siehe S. 174).

[190] Silvano Martello und Daniele Vigo. "Exact Solution of the Two-Dimensional Finite Bin Packing Problem". 44(3):388–399, März 1998. doi:10.1287/mnsc.44.3.388 (siehe S. 163, 164).

[191] Aaron Maxwell. *What are f-strings in Python and how can I use them?* Oakville, ON, Canada: Infinite Skills Inc, Juni 2017. ISBN: 978-1-4919-9486-3 (siehe S. 166).

[192] Jim Melton und Alan R. Simon. *SQL: 1999 – Understanding Relational Language Components*. The Morgan Kaufmann Series in Data Management Systems. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juni 2001. ISSN: 1046-1698. ISBN: 978-1-55860-456-8 (siehe S. 175).

References XXVI

[193] Stephan Mertens. "Exhaustive Search for Low-Autocorrelation Binary Sequences". *Journal of Physics A: Mathematical and General* 29(18):L473, Sep. 1996. Bristol, England, UK: IOP Publishing Ltd. ISSN: 0305-4470. doi:10.1088/0305-4470/29/18/005 (siehe S. 169).

[194] Peter Merz und Bernd Freisleben. "A Comparison of MAMemetic Algorithms, TSTabu Search, and ACOAnt Colonies for the QAPQuadratic Assignment Problem". In: *Congress on Evolutionary Computation (CEC'1999)*. 6.–9. Juli 1999, Washington, D.C., USA. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 1999, S. 2063–2070. doi:10.1109/CEC.1999.785529 (siehe S. 173).

[195] Zbigniew „Zbyszek" Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 1996. ISBN: 978-3-540-58090-4. doi:10.1007/978-3-662-03315-9 (siehe S. 166, 175).

[196] Burkhard Militzer, Michele Zamparelli und Dieter Beule. "Evolutionary Search for Low Autocorrelated Binary Sequences". *IEEE Transactions on Evolutionary Computation* 2(1):34–39, Apr. 1998. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/4235.728212 (siehe S. 169).

[197] Alfonsas Misevičius. "A Tabu Search Algorithm for the Quadratic Assignment Problem". *Computational Optimization and Applications* 30:95–111, Jan. 2005. London, England, UK: Springer Nature Limited. ISSN: 0926-6003. doi:10.1007/s10589-005-4562-x (siehe S. 176).

[198] Alfonsas Misevičius. "A Modified Simulated Annealing Algorithm for the Quadratic Assignment Problem". *Informatica* 14(4):497–514, Jan. 2003. Vilnius, Lithuania: Vilnius University, Institute of Data Science and Digital Technologies (DMSTI), Lithuanian Academy of Sciences. ISSN: 0868-4952. doi:10.15388/Informatica.2003.037 (siehe S. 174).

[199] Alfonsas Misevičius. *An Implementation of the Iterated Tabu Search Algorithm for the Quadratic Assignment Problem*. Working Paper. Kaunas, Lithuania: Kaunas University of Technology, 2008. See also[200] (siehe S. 146).

[200] Alfonsas Misevičius. "An Implementation of the Iterated Tabu Search Algorithm for the Quadratic Assignment Problem". *OR Spectrum: Quantitative Approaches in Management. Formerly Operations-Research-Spektrum* 34(3):665–690, Juli 2012. London, England, UK: Springer Nature Limited. ISSN: 0171-6468. doi:10.1007/s00291-011-0274-z. See also[199] (siehe S. 146, 173, 176).

[201] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Complex Adaptive Systems. Cambridge, MA, USA: MIT Press, Feb. 1998. ISBN: 978-0-262-13316-6. URL: http://boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf (besucht am 2025-08-08) (siehe S. 166, 175).

References XXVII

[202] Nenad Mladenović und Pierre Hansen. "Variable Neighborhood Search". *Computers & Operations Research* 24(11):1097–1100, Nov. 1997. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0305-0548. doi:10.1016/S0305-0548(97)00031-2 (siehe S. 179).

[203] Wai Ho Mow, Ke-Lin Du und Wei Hsiang Wu. "New Evolutionary Search for Long Low Autocorrelation Binary Sequences". *IEEE Transactions on Aerospace and Electronic Systems* 51(1):290–303, Jan. 2015. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 0018-9251. doi:10.1109/TAES.2014.130518 (siehe S. 169).

[204] Heinz Mühlenbein. "How Genetic Algorithms Really Work: Mutation and Hillclimbing". In: *Parallel Problem Solving from Nature 2 (PPSN-II)*. 28.–30. Sep. 1992, Brussels, Belgium. Hrsg. von Reinhard Männer und Bernard Manderick. Amsterdam, The Netherlands: Elsevier B.V., 1992, S. 15–26. URL: https://www.researchgate.net/publication/284805798 (besucht am 2025-08-15) (siehe S. 172).

[205] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 164).

[206] Siegfried Nijssen und Thomas Bäck. "An Analysis of the Behavior of Simplified Evolutionary Algorithms on Trap Functions". *IEEE Transactions on Evolutionary Computation* 7(1):11–22, Feb. 2003. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2002.806169 (siehe S. 176).

[207] Eugeniusz Nowicki und Czesław Smutnicki. "A Fast Taboo Search Algorithm for the Job Shop Problem". *Management Science* 42(6):783–938, Juni 1996. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0025-1909. doi:10.1287/mnsc.42.6.797 (siehe S. 176).

[208] Regina O. Obe und Leo S. Hsu. *PostgreSQL: Up and Running*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Okt. 2017. ISBN: 978-1-4919-6336-4 (siehe S. 173).

[209] Robert Orfali, Dan Harkey und Jeri Edwards. *Client/Server Survival Guide*. 3. Aufl. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 25. Jan. 1999. ISBN: 978-0-471-31615-2 (siehe S. 165).

[210] Tom Packebusch und Stephan Mertens. "Low Autocorrelation Binary Sequences". *Journal of Physics A: Mathematical and Theoretical* 49(16):165001, März 2016. Bristol, England, UK: IOP Publishing Ltd. ISSN: 1751-8121. doi:10.1088/1751-8113/49/16/165001 (siehe S. 169).

References XXVIII

[211] Panos Miltiades Pardalos, Ding-Zhu Du und Ronald Lewis Graham, Hrsg. *Handbook of Combinatorial Optimization*. 1. Aufl. Boston, MA, USA: Springer, 1998. ISBN: 978-1-4613-7987-4.

[212] Panos Miltiades Pardalos, Oleg V. Shylo und Alkis Vazacopoulos. "Solving Job Shop Scheduling Problems Utilizing the Properties of Backbone and "Big Valley"". *Computational Optimization and Applications* 47(1):61–76, Sep. 2010. London, England, UK: Springer Nature Limited. ISSN: 0926-6003. doi:10.1007/s10589-008-9206-5. An algorithm similar to SA. (Siehe S. 168, 174).

[213] Francisco Parreño, Ramón Alvarez-Valdés, José Fernando Oliveira und José Manuel Tamarit. "A Hybrid GRASP/VND Algorithm for Two- and Three-Dimensional Bin Packing". *Annals of Operations Research* 179(1):203–220, Sep. 2010. London, England, UK: Springer Nature Limited. ISSN: 0254-5330. doi:10.1007/s10479-008-0449-4 (siehe S. 163, 167, 178).

[214] Bo Peng, Zhipeng Lü und T. C. Edwin Cheng. "A Tabu Search/Path Relinking Algorithm to Solve the Job Shop Scheduling Problem". *Computers & Operations Research* 53:154–164, Jan. 2015. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0305-0548. doi:10.1016/j.cor.2014.08.006. URL: http://arxiv.org/abs/1402.5613 (besucht am 2026-09-30) (siehe S. 168, 176).

[215] Yasset Pérez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J. Eglen, Daniel S. Katz, Tom J. Pollard, Alexander Konovalov, Robert M. Flight, Kai Blin und Juan Antonio Vizcaíno. "Ten Simple Rules for Taking Advantage of Git and GitHub". *PLOS Computational Biology* 12(7), 14. Juli 2016. San Francisco, CA, USA: Public Library of Science (PLOS). ISSN: 1553-7358. doi:10.1371/JOURNAL.PCBI.1004947 (siehe S. 167).

[216] Tim Peters. "Algorithms – Introduction". In: *Python Cookbook*. Hrsg. von David Ascher. 1. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Juli 2002. Kap. 17. ISBN: 978-0-596-00167-4. URL: https://www.oreilly.com/library/view/python-cookbook/0596001673/ch17.html (besucht am 2024-11-07) (siehe S. 73–93).

[217] Ferdinando Pezzella und Emanuela Merelli. "A Tabu Search Method Guided by Shifting Bottleneck for the Job Shop Scheduling Problem". *European Journal of Operational Research* 120(2):297–310, Jan. 2000. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(99)00158-7. URL: https://www2.cs.sfu.ca/CourseCentral/827/havens/papers/topic%5C%2310(JobShop)/Tabu%5C%20With%5C%20Shifting.pdf (besucht am 2026-09-30) (siehe S. 176).

References XXIX

[218] Martin Pincus. "Letter to the Editor – A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems". *Operations Research* 18(6):1225–1228, Nov.–Dez. 1970. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0030-364X. doi:10.1287/opre.18.6.1225 (siehe S. 174).

[219] Marcelo Pinto, Cristóvão Silva, Matthias Thürer und Samuel Moniz. "Survey in Operations Research and Management Science. Nesting and Scheduling Optimization of Additive Manufacturing Systems: Mapping the Territory". *Computers & Operations Research* 165(106592), Feb. 2024. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0305-0548. doi:10.1016/j.cor.2024.106592. Mentions some scheduling tasks of Additive Manufacturing jobs can be viewed as instance of the 2BP|R|F. (Siehe S. 163).

[220] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 173).

[221] S. D. Prestwich. *Improved Branch-and-Bound for Low Autocorrelation Binary Sequences*. arXiv.org: Computing Research Repository (CoRR) abs/1305.6187. Ithaca, NY, USA: Cornell Universiy Library, 3. Apr. 2024. doi:10.48550/arXiv.1305.6187. URL: https://arxiv.org/abs/1305.6187 (besucht am 2013-07-23). arXiv:1305.6187v2 [cs.AI] 23 Jul 2013 (siehe S. 169).

[222] *Programming Languages – C, Working Document of SC22/WG14*. International Standard ISO/IEC9899:2017 C17 Ballot N2176. Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Nov. 2017. URL: https://files.lhmouse.com/standards/ISO%20C%20N2176.pdf (besucht am 2024-06-29) (siehe S. 165).

[223] Sanatan Rai und George Vairaktarakis. "NP-Complete Problems and Proof Methodology". In: *Encyclopedia of Optimization*. Hrsg. von Christodoulos A. Floudas und Panos Miltiades Pardalos. 2. Aufl. Boston, MA, USA: Springer, Sep. 2008, S. 2675–2682. ISBN: 978-0-387-74758-3. doi:10.1007/978-0-387-74759-0_462 (siehe S. 181).

[224] Soraya B. Rana und L. Darrell Whitley. "Genetic Algorithm Behavior in the MAXSAT Domain". In: *5th International Conference on Parallel Problem Solving from Nature (PPSN V)*. 27.–30. Sep. 1998, Amsterdam, The Netherlands. Hrsg. von Ágoston Endre Eiben, Thomas Bäck, Marc Schoenauer und Hans-Paul Schwefel. Lecture Notes in Computer Science (LNCS). 1998, S. 785–794. ISSN: 0302-9743. ISBN: 978-3-540-65078-2. doi:10.1007/BFb0056920 (siehe S. 171).

[225] Abhishek Ratan, Eric Chou, Pradeeban Kathiravelu und Dr. M.O. Faruque Sarker. *Python Network Programming*. Birmingham, England, UK: Packt Publishing Ltd, Jan. 2019. ISBN: 978-1-78883-546-6 (siehe S. 165).

References XXX

[226] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: 978-1-78398-154-0 (siehe S. 170).

[227] Mike Reichardt, Michael Gundall und Hans D. Schotten. "Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients". In: *47th Annual Conference of the IEEE Industrial Electronics Society (IECON'2021*. 13.–15. Okt. 2021, Toronto, ON, Canada. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2021, S. 1–8. ISSN: 2577-1647. ISBN: 978-1-6654-3554-3. doi:10.1109/IECON48115.2021.9589382 (siehe S. 171).

[228] Gerhard Reinelt. "TSPLIB – A Traveling Salesman Problem Library". *ORSA Journal on Computing* 3(4):376–384, Herbst 1991. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0899-1499. doi:10.1287/ijoc.3.4.376 (siehe S. 176, 177).

[229] Gerhard Reinelt. *TSPLIB95*. Techn. Ber. Heidelberg, Baden-Württemberg, Germany: Universität Heidelberg, Institut für Angewandte Mathematik, 1995–28. Apr. 2016. URL: https://github.com/shredderzwj/TSPLIB (besucht am 2026-09-25). The original Uniform Resource Locator (URL) of the repository has become defunct in 2026. Therefore, we provide the repository listed as main URL which contains the data. Additionally, you may also check https://github.com/thomasWeise/tspSuite of our own project *TSPSuite*, which contains all the TSPLib instances as well as Java code to load them. (Siehe S. 176, 177).

[230] Mark Richards und Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, Inc., Jan. 2020. ISBN: 978-1-4920-4345-4 (siehe S. 165).

[231] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*. Bd. 35 der Reihe Forschungsergebnisse zur Informatik. Hamburg, Germany: Verlag Dr. Kovač, 1997. ISSN: 1435-6260. ISBN: 978-3-86064-554-3 (siehe S. 169).

[232] Stuart J. Russell und Peter Norvig. *Artificial Intelligence: A Modern Approach (AIMA)*. 4. Aufl. Hoboken, NJ, USA: Pearson Education, Inc. ISBN: 978-1-292-40113-3. URL: https://aima.cs.berkeley.edu (besucht am 2024-06-27) (siehe S. 163).

[233] Yeonhee Ryou, Sangwoo Joh, Joonmo Yang, Sujin Kim und Youil Kim. "Code Understanding Linter to Detect Variable Misuse". In: *37th IEEE/ACM International Conference on Automated Software Engineering (ASE'2022)*. 10.–14. Okt. 2022, Rochester, MI, USA. New York, NY, USA: Association for Computing Machinery (ACM), 2022, 133:1–133:5. ISBN: 978-1-4503-9475-8. doi:10.1145/3551349.3559497 (siehe S. 169).

References XXXI

[234] Sartaj Sahni und Teofilo Francisco Gonzalez Arce. "P-complete Approximation Problems". *Journal of the ACM (JACM)* 23(3):555–565, Juli 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0004-5411. doi:10.1145/321958.321975. URL: https://sites.cs.ucsb.edu/~teo/papers/JACM-PCom.pdf (besucht am 2026-02-21) (siehe S. 173).

[235] Guntram Scheithauer. *Introduction to Cutting and Packing Optimization: Problems, Modeling Approaches, Solution Methods*. Bd. 263 der Reihe International Series in Operations Research & Management Science (ISOR). Cham, Switzerland: Springer, 2018. ISSN: 0884-8289. ISBN: 978-3-319-64402-8. doi:10.1007/978-3-319-64403-5 (siehe S. 164).

[236] Shai Shalev-Shwartz und Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge, England, UK: Cambridge University Press (CUP), Juli 2014. ISBN: 978-1-107-05713-5. URL: http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning (besucht am 2024-06-27) (siehe S. 171).

[237] Guoyong Shi, Hitoshi Iima und Nobuo Sannomiya. "New Encoding Scheme for Solving Job Shop Problems by Genetic Algorithm". In: *35th IEEE Conference on Decision and Control (CDC'96)*. Bd. 4. 11.–13. Dez. 1996. Kōbe, Hyōgo-ken, Japan: Institute of Electrical and Electronics Engineers (IEEE), 1997, S. 4395–4400. ISBN: 978-0-7803-3590-5. doi:10.1109/CDC.1996.577484. URL: https://www.researchgate.net/publication/224238934 (besucht am 2026-09-30). Proposes a permutations-with-repetitions based encoding for the JSSP. (Siehe S. 136, 168, 180).

[238] Oleg V. Shylo. *Job Shop Scheduling (Personal Homepage)*. Knoxville, TN, USA: Oleg V. Shylo, Aug. 2019. URL: http://optimizizer.com/jobshop.php (besucht am 2026-09-30). Contains many JSSP benchmark instances. (Siehe S. 151, 168).

[239] Oleg V. Shylo und Hesam Shams. *Boosting Binary Optimization via Binary Classification: A Case Study of Job Shop Scheduling*. arXiv.org: Computing Research Repository (CoRR) abs/1808.10813. Proposes a TS using regression to adapt the tabu tenures for each variable of the solution vector of a JSSP separately. Many results are available in the GitHub repository https://github.com/quasiquasar/gta-jobshop-data; other results are published Shylo's website http://optimizizer.com/jobshop.php. Ithaca, NY, USA: Cornell Universiy Library, 31. Aug. 2018. doi:10.48550/arXiv.1808.10813. URL: https://arxiv.org/abs/1808.10813 (besucht am 2026-09-30) (siehe S. 168, 176).

[240] Ellen Siever, Stephen Figgins, Robert Love und Arnold Robbins. *Linux in a Nutshell*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2009. ISBN: 978-0-596-15448-6 (siehe S. 170).

[241] Anna Skoulikari. *Learning Git*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mai 2023. ISBN: 978-1-0981-3391-7 (siehe S. 167).

References XXXII

[242] Eric V. „ericvsmith“ Smith. *Literal String Interpolation*. Python Enhancement Proposal (PEP) 498. Beaverton, OR, USA: Python Software Foundation (PSF), 6. Nov. 2016–9. Sep. 2023. URL: https://peps.python.org/pep-0498 (besucht am 2024-07-25) (siehe S. 166).

[243] John Miles Smith und Philip Yen-Tang Chang. “Optimizing the Performance of a Relational Algebra Database Interface”. *Communications of the ACM (CACM)* 18(10):568–579, Okt. 1975. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/361020.361025 (siehe S. 174).

[244] Hossein M. Soroush. “Scheduling in Stochastic Bicriteria Single Machine Systems with Set-up Times”. *International Journal of Planning and Scheduling* 1(1–2):109–145, Jan. 2011. Geneva, Switzerland: Inderscience Publishers. ISSN: 2044-494X. doi:10.1504/IJPS.2011.044605. A scheduling problem is phrased as QAP. (Siehe S. 173).

[245] “SQL Commands”. In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Part VI. Reference. URL: https://www.postgresql.org/docs/17/sql-commands.html (besucht am 2025-02-25) (siehe S. 175).

[246] Leon Steinberg. “The Backboard Wiring Problem: A Placement Algorithm”. *SIAM Review* 3(1):37–50, Jan. 1961. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics Publications (SIAM). ISSN: 0036-1445. doi:10.1137/1003003. The problem is modeled as a QAP (siehe S. 173).

[247] Ryan K. Stephens und Ronald R. Plew. *Sams Teach Yourself SQL in 21 Days*. 4. Aufl. Sams Teach Yourself. Indianapolis, IN, USA: SAMS Technical Publishing und Hoboken, NJ, USA: Pearson Education, Inc., Okt. 2002. ISBN: 978-0-672-32451-2 (siehe S. 152, 175).

[248] Ryan K. Stephens, Ronald R. Plew, Bryan Morgan und Jeff Perkins. *SQL in 21 Tagen. Die Datenbank-Abfragesprache SQL vollständig erklärt (in 14/21 Tagen)*. 6. Aufl. Burgthann, Bayern, Germany: Markt+Technik Verlag GmbH, Feb. 1998. ISBN: 978-3-8272-2020-2. Translation of [247] (siehe S. 175).

[249] Robert H. Storer, S. David Wu und Renzo Vaccari. “New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling”. *Management Science* 38(10):1495–1509, Okt. 1992. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0025-1909. doi:10.1287/mnsc.38.10.1495 (siehe S. 168).

References XXXIII

[250] Thomas Strecker und Leonhard Hennig. "Automatic Layouting of Personalized Newspaper Pages". In: *Operations Research Proceedings 2008: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR)*. 3.–5. Sep. 2008, Augsburg, Bayern, Germany: University of Augsburg. Hrsg. von Bernhard Fleischmann, Karl-Heinz Borgwardt, Robert Klein und Axel Tuma. Bd. 2008 der Reihe Operations Research Proceedings (ORP), Gesellschaft für Operations Research e.V., (GOR). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2009, S. 469–474. ISSN: 0721-5924. doi:10.1007/978-3-642-00142-0_76 (siehe S. 163).

[251] Thomas Stützle. "Iterated Local Search for the Quadratic Assignment Problem". *European Journal of Operational Research* 174(3):1519–1539, Nov. 2006. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2005.01.066 (siehe S. 167, 173).

[252] Dirk Sudholt. "Crossover Speeds up Building-Block Assembly". In: *14th Genetic and Evolutionary Computation Conference (GECCO'2012)*. 7.–11. Juli 2012, Philadelphia, PA, USA. Hrsg. von Terence Soule und Jason Hall Moore. New York, NY, USA: Association for Computing Machinery (ACM), 2012, S. 689–702. ISBN: 978-1-4503-1177-9. doi:10.1145/2330163.2330260. This paper is on the Greedy $(2+1)$ Genetic Algorithm (Greedy $(2+1)$ GA), see also[253] (siehe S. 153, 167).

[253] Dirk Sudholt. "How Crossover Speeds up Building Block Assembly in Genetic Algorithms". *Evolutionary Computation* 25(2):237–274, 2017. Cambridge, MA, USA: MIT Press. ISSN: 1063-6560. doi:10.1162/EVCO_a_00171. This paper is on the Greedy $(2+1)$ GA, see also[252] (siehe S. 153, 167).

[254] Éric D. Taillard. "Benchmarks for Basic Scheduling Problems". *European Journal of Operational Research* 64(2):278–285, Jan. 1993. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/0377-2217(93)90182-M. URL: https://arodes.hes-so.ch/record/8224/files/Preprint.pdf (besucht am 2026-02-20) (siehe S. 168).

[255] Éric D. Taillard. "Comparison of Iterative Searches for the Quadratic Assignment Problem". *Location Science. Now incorporated into Computers & Operations Research* 3(2):87–105, Aug. 1995. Amsterdam, The Netherlands: Elsevier B.V. doi:10.1016/0966-8349(95)00008-6 (siehe S. 176).

[256] Éric D. Taillard. "Robust Taboo Search for the Quadratic Assignment Problem". *Parallel Computing* 17(4–5):443–455, Juli 1991. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0167-8191. doi:10.1016/S0167-8191(05)80147-4 (siehe S. 134, 154, 173, 176).

References XXXIV

[257] Éric D. Taillard und Luca M. Gambardella. *Adaptive Memories for the Quadratic Assignment Problem*. Techn. Ber. IDSIA-87-97. Lugano, Switzerland: Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale (IDSIA), 10. Sep. 1997. URL: https://mistic.iict-heig-vd.ch/taillard/articles.dir/TaillardGambardella1997.pdf (besucht am 2026-09-28). A MA using TS from [256] as well as a ACO-based approach refining its solutions with a very simple SLS are presented. (Siehe S. 173).

[258] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 174, 175).

[259] Alkin Tezuysal und Ibrar Ahmed. *Database Design and Modeling with PostgreSQL and MySQL*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2024. ISBN: 978-1-80323-347-5 (siehe S. 171, 173).

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

[261] ."Literals". In: *Static Typing with Python*. Hrsg. von The Python Typing Team. Beaverton, OR, USA: Python Software Foundation (PSF), 2021. URL: https://typing.python.org/en/latest/spec/literal.html (besucht am 2025-08-29) (siehe S. 170).

[262] *The Unicode Standard, Version 15.1: Archived Code Charts*. South San Francisco, CA, USA: The Unicode Consortium, 25. Aug. 2023. URL: https://www.unicode.org/Public/15.1.0/charts/CodeCharts.pdf (besucht am 2024-07-26) (siehe S. 178).

[263] Dirk Thierens, David Edward Goldberg und Angela Guimarães Pereira. "Domino Convergence, Drift, and the Temporal-Salience Structure of Problems". In: S. 535–540. doi:10.1109/ICEC.1998.700085 (siehe S. 162).

[264] Sarah Louise Thomson, Gabriela Ochoa, Daan van den Berg, Tianyu Liang (梁天宇) und Thomas Weise (汤卫思). "Entropy, Search Trajectories, and Explainability for Frequency Fitness Assignment". In: *Parallel Problem Solving from Nature (PPSN XVIII)*. Bd. 1. 14.–18. Sep. 2024, Hagenberg, Mühlkreis, Austria. Hrsg. von Michael Affenzeller, Stephan M. Winkler, Anna V. Kononova, Heike Trautmann, Tea Tušar, Penousal Machado und Thomas Bäck. Bd. 15148 der Reihe Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer. ISSN: 0302-9743. ISBN: 978-3-031-70054-5. doi:10.1007/978-3-031-70055-2_23. Explores randomized local search (RLS), RLS with FFA (FRLS), and SA on the QAP. (Siehe S. 166, 171).

References XXXV

[265] Ulrich Wilhelm Thonemann und Andreas Bölte. *An Improved Simulated Annealing Algorithm for the Quadratic Assignment Problem*. Working Paper. Paderborn, Nordrhein-Westfalen, Germany: School of Business, Department of Production and Operations Research, University of Paderborn, 1994 (siehe S. 174).

[266] "`timeit` – Measure Execution Time of Small Code Snippets". In: *Python 3 Documentation*. *The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library/timeit.html (besucht am 2024-11-07) (siehe S. 73–76, 87–92).

[267] Marco Tomassini. "Complex Networks Analysis of the Energy Landscape of the Low Autocorrelation Binary Sequences Problem". *Physica A: Statistical Mechanics and its Applications* 577:126089, 1. Sep. 2021. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0378-4371. doi:10.1016/j.physa.2021.126089. Lists the LABS as $\mathcal{NP}$-hard. (Siehe S. 169).

[268] Linus Torvalds. "The Linux Edge". *Communications of the ACM (CACM)* 42(4):38–39, Apr. 1999. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/299157.299165 (siehe S. 170).

[269] Mariot Tsitoara. *Beginning Git and GitHub: Version Control, Project Management and Teamwork for the New Developer*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0215-7 (siehe S. 167, 178).

[270] *Unicode® 15.1.0*. South San Francisco, CA, USA: The Unicode Consortium, 12. Sep. 2023. ISBN: 978-1-936213-33-7. URL: https://www.unicode.org/versions/Unicode15.1.0 (besucht am 2024-07-26) (siehe S. 178).

[271] Daan van den Berg, Florian Braam, Mark Moes, Emiel Suilen und Sandjai Bhulai. "Almost Squares in Almost Squares: Solving the Final Instance". In: *5th International Conference on Data Analytics (DATA ANALYTICS'2016)*. 9.–13. Okt. 2016, Venice, Venezia, Veneto, Italy. Hrsg. von Sandjai Bhulai und Ivana Semanjski. Wilmington, DE, USA: International Academy, Research, and Industry Association (IARIA), 2016, S. 69–74. ISSN: 2308-4464. ISBN: 978-1-61208-510-4. URL: https://math.vu.nl/~sbhulai/publications/data_analytics2016b.pdf (besucht am 2026-10-01). This poses a very interesting 2BP|R|F problem, where „almost squars," i.e., rectangles with sizes $(i, i+1)$, must be placed in an almost square. (Siehe S. 163, 164).

References XXXVI

[272] Clarissa Van Hoyweghen, David Edward Goldberg und Bart Naudts. "From Twomax To The Ising Model: Easy And Hard Symmetrical Problems". In: *4th Genetic and Evolutionary Computation Conference (GECCO'2002)*. 9.–13. Juli 2002, New York, NY, USA. Hrsg. von William B. Langdon, Erick Cantú-Paz, Keith E. Mathias, Rajkumar Roy, David Davis, Riccardo Poli, Karthik Balakrishnan, Vasant G. Honavar, Günter Rudolph, Joachim Wegener, Larry Bull, Mitchell A. Potter, Alan C. Schultz, Julian F. Miller, Edmund K. Burke und Natasa Jonoska. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, 2002, S. 626–633. ISBN: 978-1-55860-878-8 (siehe S. 177).

[273] Guido van Rossum und Łukasz Langa. *Type Hints*. Python Enhancement Proposal (PEP) 484. Beaverton, OR, USA: Python Software Foundation (PSF), 29. Sep. 2014. URL: https://peps.python.org/pep-0484 (besucht am 2024-08-22) (siehe S. 177).

[274] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 170).

[275] Dieter Vandesande, Christoph Jabs, Jeremias Berg, Johannes Fichte, George Katsirelos, Ruben Martins, André Schidler und Dominik Schreiber. *MaxSAT Evaluation 2026: Solvers, Benchmarks & Results*. v1. Geneva, Switzerland: zenodo (hosted by European Organization for Nuclear Research (CERN)), 14. Juli 2026. doi:10.5281/zenodo.21358014. See also https://maxsat-evaluations.github.io. (Siehe S. 171).

[276] Kristian Verduin, Sarah Louise Thomson und Daan van den Berg. "Too Constrained for Genetic Algorithms. Too Hard for Evolutionary Computing. The Traveling Tournament Problem". In: *15th International Joint Conference on Computational Intelligence (IJCCI'23)*. 13.–15. Nov. 2023, Rome, Italy. Hrsg. von Niki van Stein, Francesco Marcelloni, H. K. Lam, Marie Cottrell und Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2023, S. 246–257. ISSN: 2184-3236. ISBN: 978-989-758-674-3. doi:10.5220/0012192100003595 (siehe S. 177).

[277] Petr Vilím, Philippe Laborie und Paul Shaw. "Failure-Directed Search for Constraint-Based Scheduling". In: *12th International Conference on the Integration of AI and OR Techniques in Constraint Programming (CPAIOR'2015)*. Hrsg. von Laurent Michel. Bd. 9075 der Reihe Theoretical Computer Science and General Issues (LNTCS), sub-series of Lecture Notes in Computer Science (LNCS). Barcelona, Spain: Springer, 2015, S. 437–453. ISSN: 2512-2010. ISBN: 978-3-319-18007-6. doi:10.1007/978-3-319-18008-3_30. URL: https://www.researchgate.net/publication/280223626 (besucht am 2026-09-30) (siehe S. 168).

References XXXVII

[278] Barry Warsaw. *List Comprehensions*. Python Enhancement Proposal (PEP) 202. Beaverton, OR, USA: Python Software Foundation (PSF), 13. Juli 2000. URL: https://peps.python.org/pep-0202 (besucht am 2024-11-08) (siehe S. 5–10, 12–14).

[279] Thomas Weise (汤卫思). *Databases*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), 2025. URL: https://thomasweise.github.io/databases (besucht am 2025-01-05) (siehe S. 165, 174).

[280] Thomas Weise (汤卫思). *Global Optimization Algorithms – Theory and Application*. self-published, 2009. URL: https://www.researchgate.net/publication/200622167 (besucht am 2025-07-25) (siehe S. 166, 170, 171, 174, 175).

[281] Thomas Weise (汤卫思). *jsspInstancesAndResults: Results, Data, and Instances of the Job Shop Scheduling Problem*. San Francisco, CA, USA: GitHub Inc, 2019–2020. URL: https://github.com/thomasWeise/jsspInstancesAndResults (besucht am 2025-08-20) (siehe S. 168).

[282] Thomas Weise (汤卫思). *Programming with C*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), 2026. URL: https://thomasweise.github.io/programmingWithC (besucht am 2026-05-10) (siehe S. 165).

[283] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), 2024–2026. URL: https://thomasweise.github.io/programmingWithPython (besucht am 2025-01-05) (siehe S. 171, 173, 174).

[284] Thomas Weise (汤卫思), Raymond Chiong, Jörg Lässig, Ke Tang (唐珂), Shigeyoshi Tsutsui, Wenxiang Chen (陈文祥), Zbigniew „Zbyszek“ Michalewicz und Xin Yao (姚新). “Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem”. *IEEE Computational Intelligence Magazine (CIM)* 9(3):40–52, Aug. 2014. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1556-603X. doi:10.1109/MCI.2014.2326101. Applies a multitude of algorithms including SLSes, EAs, MAs, ACO, and EDAs to the TSPLib instances. (Siehe S. 63–76, 163, 176, 179).

[285] Thomas Weise (汤卫思), Xinlu Li (李新路), Yan Chen (陈岩) und Zhize Wu (吴志泽). “Solving Job Shop Scheduling Problems without using a Bias for Good Solutions”. In: *23rd Genetic and Evolutionary Computation Conference (GECCO'2021), Companion Volume*. 10.–14. Juli 2021, Lille, France. Hrsg. von Krzysztof Krawiec. New York, NY, USA: Association for Computing Machinery (ACM), 2021, S. 1459–1466. ISBN: 978-1-4503-8351-6. doi:10.1145/3449726.3463124. Applies Frequency Fitness Assignment (FFA) to the JSSP. (Siehe S. 180).

References XXXVIII

[286] Thomas Weise (汤卫思), Mingxu Wan (万明绪), Pu Wang (王璞), Ke Tang (唐珂), Alexandre Devert und Xin Yao (姚新). "Frequency Fitness Assignment". *IEEE Transactions on Evolutionary Computation* 18(2):226–243, Apr. 2014. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2013.2251885. The article introducing FFA. (Siehe S. 166).

[287] Thomas Weise (汤卫思) und Zhize Wu (吴志泽). "Replicable Self-Documenting Experiments with Arbitrary Search Spaces and Algorithms". In: *25th Conference on Genetic and Evolutionary Computation (GECCO'2023), Companion Volume*. 15.–19. Juli 2023, Lisbon, Portugal. Hrsg. von Sara Silva und Luís Paquete. New York, NY, USA: Association for Computing Machinery (ACM), 2023, S. 1891–1899. ISBN: 979-8-4007-0120-7. doi:10.1145/3583133.3596306. Introduces the experiment execution and data collection facilities of moptipy. (Siehe S. 171).

[288] Thomas Weise (汤卫思), Zhize Wu (吴志泽), Xinlu Li (李新路) und Yan Chen (陈岩). "Frequency Fitness Assignment: Making Optimization Algorithms Invariant under Bijective Transformations of the Objective Function Value". *IEEE Transactions on Evolutionary Computation* 25(2):307–319, Apr. 2021. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2020.3032090 (siehe S. 162, 166, 168, 169, 172, 176, 177).

[289] Thomas Weise (汤卫思), Zhize Wu (吴志泽), Xinlu Li (李新路), Yan Chen (陈岩) und Jörg Lässig. "Frequency Fitness Assignment: Optimization without Bias for Good Solutions can be Efficient". *IEEE Transactions on Evolutionary Computation* 27(4):980–992, 2023. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2022.3191698. Applies many FFA-based algorithm variants to problems such as OneMax, LeadingOnes, Linear Harmonic Function (LinHarm), TwoMax, Trap, Jump, Plateau, Ising Model, and N-Queens. (Siehe S. 162, 166, 169, 176, 177).

[290] Thomas Weise (汤卫思), Yuezhong Wu (吴越钟), Raymond Chiong, Ke Tang (唐珂) und Jörg Lässig. "Global versus Local Search: The Impact of Population Sizes on Evolutionary Algorithm Performance". *Journal of Global Optimization* 66(3):511–534, Feb. 2016. London, England, UK: Springer Nature Limited. ISSN: 0925-5001. doi:10.1007/s10898-016-0417-5 (siehe S. 163, 176).

[291] *What does PDF mean?* San Jose, CA, USA: Adobe Systems Incorporated, 2024. URL: https://www.adobe.com/acrobat/about-adobe-pdf.html (besucht am 2024-12-12) (siehe S. 172).

[292] *What is a Relational Database?* Armonk, NY, USA: International Business Machines Corporation (IBM), 20. Okt. 2021–12. Dez. 2024. URL: https://www.ibm.com/think/topics/relational-databases (besucht am 2025-01-05) (siehe S. 174).

References XXXIX

[293] L. Darrell Whitley. "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best". In: *3rd International Conference on Genetic Algorithms (ICGA'1989)*. Juni 1989, Fairfax, VA, USA: George Mason University. Hrsg. von J. David Schaffer. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, 1989, S. 116–123. ISBN: 978-1-55860-066-9. URL: https://www.researchgate.net/publication/2527551 (besucht am 2025-08-08) (siehe S. 169, 175).

[294] Ulf Michael „Monty" Widenius, David Axmark und Uppsala, Sweden: MySQL AB. *MySQL Reference Manual – Documentation from the Source*. Sebastopol, CA, USA: O'Reilly Media, Inc., 9. Juli 2002. ISBN: 978-0-596-00265-7 (siehe S. 171).

[295] James M. Wilson. "Gantt Charts: A Centenary Appreciation". *European Journal of Operational Research* 149(2):430–437, Sep. 2003. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(02)00769-5 (siehe S. 166).

[296] L. Wong und Lai-Soon Lee. "Heuristic Placement Routines for Two-Dimensional Bin Packing Problem". 5(4):334–341, Dez. 2009. doi:/10.3844/jmssp.2009.334.341. URL: https://www.thescipub.com/pdf/jmssp.2009.334.341 (besucht am 2026-10-02). A heuristic for the 2BP|R|F and the 2BP|O|F (siehe S. 125).

[297] CAO Xiang (曹翔), Zhize Wu (吴志泽), Daan van den Berg und Thomas Weise (汤卫思). "Randomized Local Search vs. NSGA-II vs. Frequency Fitness Assignment on The Traveling Tournament Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. 20.–22. Nov. 2024, Porto, Portugal. Hrsg. von Francesco Marcelloni, Kurosh Madani, Niki van Stein und Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, S. 38–49. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012891500003837 (siehe S. 171, 177, 180).

[298] Zhou Xu und Chung-Yee Lee. "New Lower Bound and Exact Method for the Continuous Berth Allocation Problem". *Operations Research* 66(3):ii–iv, 597–892, Mai–Juni 2018. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0030-364X. doi:10.1287/opre.2017.1687. URL: https://ira.lib.polyu.edu.hk/bitstream/10397/78451/1/Xu_New_Lower_Bound.pdf (besucht am 2026-10-01) (siehe S. 164).

[299] Takeshi Yamada und Ryohei Nakano. "A Genetic Algorithm Applicable to Large-Scale Job-Shop Instances". In: *Parallel Problem Solving from Nature 2 (PPSN-II)*. 28.–30. Sep. 1992, Brussels, Belgium. Hrsg. von Reinhard Männer und Bernard Manderick. Amsterdam, The Netherlands: Elsevier B.V., 1992, S. 281–290. URL: https://www.researchgate.net/publication/220701684 (besucht am 2026-09-30) (siehe S. 168).

References XL

[300] Kinza Yasar und Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., Juni 2024. URL: https://www.techtarget.com/searchdatamanagement/definition/database-management-system (besucht am 2025-01-11) (siehe S. 165).

[301] François Yergeau. *UTF-8, A Transformation Format of ISO 10646*. Request for Comments (RFC) 3629. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Nov. 2003. URL: https://www.ietf.org/rfc/rfc3629.txt (besucht am 2025-02-05). See Unicode and[137] (siehe S. 178).

[302] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 164).

[303] Jiazheng Zeng (曾嘉政), Thomas Weise (汤卫思), Zhize Wu (吴志泽) und Markus Wagner. "When Frequency Fitness Assignment Fails: Trapped States in Frequency-Guided Local Search". In: *28th Genetic and Evolutionary Computation Conference (GECCO'2026)*. 13.–17. Juli 2026, San José, Costa Rica. Hrsg. von Leonardo Trujillo und Ting Hu. New York, NY, USA: Association for Computing Machinery (ACM), 2026, S. 233–242. ISBN: 979-8-4007-2487-9. doi:10.1145/3795095.3805098. Explores $(1+1)$ evolutionary algorithm ($(1+1)$ EA), $(1+1)$ EA with FFA ($(1+1)$ FEA), RLS, and FRLS as well as hybrid algorithms on a variety of discrete benchmarks such as OneMax, LeadingOnes, LinHarm, Binary Integer (BinInt), TwoMax, Trap, Jump, Plateau, Ising Model, N-Queens and also applies them to the search for Low-Autocorrelation Binary Sequences (LABS). (Siehe S. 162, 166, 168, 169, 172–174, 176, 177).

[304] Chaoyong Zhang, Xinyu Shao, Yunqing Rao und Haobo Qiu. "Some New Results on Tabu Search Algorithm Applied to the Job-Shop Scheduling Problem". In: *Tabu Search*. Hrsg. von Wassim Jaziri. London, England, UK: IntechOpen Limited, 1. Aug. 2008. Kap. 8. ISBN: 978-3-902613-34-9. doi:10.5772/5593. URL: https://www.intechopen.com/chapters/4596 (besucht am 2026-09-30) (siehe S. 168, 176).

[305] Rui Zhao (赵睿), Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Daan van den Berg, Matthias Thürer und Thomas Weise (汤卫思). "Randomized Local Search on the 2D Rectangular Bin Packing Problem with Item Rotation". In: *26th Genetic and Evolutionary Computation Conference (GECCO'2024)*. 14.–18. Juli 2024, Melbourne, VIC, Australia. Hrsg. von Xiaodong Li und Julia Handl. New York, NY, USA: Association for Computing Machinery (ACM), 2024, S. 235–238. ISBN: 979-8-4007-0494-9. doi:10.1145/3638530.3654139 (siehe S. 163, 167, 171).

References XLI

[306] Rui Zhao (赵睿), Zhize Wu (吴志泽), Daan van den Berg, Matthias Thürer, Tianyu Liang (梁天宇), Ming Tan (檀明) und Thomas Weise (汤卫思). "Randomized Local Search for Two-Dimensional Bin Packing and a Negative Result for Frequency Fitness Assignment". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. 20.–22. Nov. 2024, Porto, Portugal. Hrsg. von Francesco Marcelloni, Kurosh Madani, Niki van Stein und Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, S. 15–26. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888500003837 (siehe S. 163, 164, 167, 171).

Glossary (in English) I



$(1+1)$ EA The $(1+1)$ EA is a local search algorithm that retains the best solution x_c discovered so far during the search[49,87]. In each step, it applies a unary search operator to this best-so-far solution x_c and derives a new solution x_n. If the new solution x_n is *better or equally good* when compared with x_c, i.e., not worse, then it replaces it, i.e., is stored as the new x_c. If the search space are bit strings of length n, then the $(1+1)$ EA uses a unary search operator that flips each bit independently with probability m/n, where usually $m = 1$. This operator is the main difference to randomized local search (RLS). The $(1+1)$ EA is a special case of the $(\mu+\lambda)$ evolutionary algorithm ($(\mu+\lambda)$ EA) where $\mu = \lambda = 1$.

$(1+1)$ FEA The $(1+1)$ EA with FFA plugged in [288, 289, 303]. On the discrete bit-string domains, this algorithm has a very specific structure: Its unary search operator flips each bit with probability $1/n$, where n is the dimension of the search space, distinguishing it from the unary operator in FRLS which flips exactly one bit. On other domains, such as permutations, the term is used synonymously with FRLS.

$(\mu+\lambda)$ EA The $(\mu+\lambda)$ EA is an evolutionary algorithm (EA) where, in each generation, λ offspring solutions are generated from the current population of μ parent solutions. The offspring and parent populations are merged, yielding $\mu+\lambda$ solutions, from which then the best μ solutions are ratained to form the parent population of the next generation. If the search space is the bit strings of length n, then this algorithm usually applies a mutation operator flipping each bit independently with probability $1/n$.

BinInt The *Binary Integer Problem* is a benchmark problem for bit-string based black-box search[263,303]. It is very similar to OneMax: It tries to maximize the number of 1 bits in the bit string of a chosen dimension n. Different from OneMax, where each bit has the same weight in the objective function, the bits of BinInt have a exponentially increasing weight: The bit string is treated as a binary-encoded integer whose value should be maximized. The objective function $f(x)$ of the minimization variant of this problem can be defined as $f(x) = 2^n - 1 - \sum_{i=1}^{n} 2^{i-1} \cdot x_i$. The problem is unimodal, i.e., has only a single global optimum and no local optimum. It is fully separable, i.e., we can optimize each variable separately.

LinHarm The Linear Harmonic Function is a benchmark problem for bit-string based black-box search[81,87,303]. It is very similar to OneMax: It tries to maximize the number of 1 bits in the bit string of a chosen dimension n. Different from OneMax, where each bit has the same weight in the objective function, the bits of LinHarm have a linear increasing weight. The objective function $f(x)$ of the minimization variant of this problem can be defined as $f(x) = \frac{1}{2}n(n+1) - \sum_{i=1}^{n} i \cdot x_i$. The problem is unimodal, i.e., has only a single global optimum and no local optimum. It is fully separable, i.e., we can optimize each variable separately. The $(1+1)$ EA can solve this problem with an expected runtime in $\Theta(n \ln n)$[81,87].

Glossary (in English) II

TSP Suite is a open source benchmarking framework for the TSP written in Java[284]. It uses the instances of the TSPLib and offers both automated experiment execution as well as several metrics to measure runtime. It is available at https://thomasweise.github.io/tspSuite and has been used in many research works, such as[290]. It can be viewed as a predecessor of moptipy.

2BP In the two-dimensional Bin Packing Problem (BP), the goal is to pack rectangles into rectangular bins without overlap and such that their edges are parallel to the edges of the bins[138,174,176]. The 2BP is strongly $\mathcal{NP}$-hard[176]. The variants of this problem include the 2BP|O|F, 2BP|R|F, 2BP|O|G, and 2BP|R|G[175,176]. Different objective function for the 2BP|O|F and 2BP|R|F are presented in [305]. 2BPs have many practical applications, including cutting (wood, glas)[176], packing (transportation, warehouse)[176], and newspaper page layout[250].

2BP|O|F In the oriented 2BP with free cuts, the rectangles to pack have fixed orientations, i.e., cannot be rotated, but can otherwise be arranged freely[175,176,190,306]. Permutation-based search spaces can be used for this problem by using the Improved Bottom Left (IBL) encoding. The state-of-the-art (SOTA) for this problem comprises[27,32,58,116,213,306].

2BP|O|G In the oriented 2BP with guillotine cuts, the rectangles to pack have fixed orientations, i.e., cannot be rotated, and must be arranged such that they can be extracted by edge-to-edge cuts[175,176].

2BP|R|F In this variant of the 2BP with free cuts, the rectangles to pack can be rotated by 90° if need be and can freely be arranged[23,175,176,305,306]. Besides packing and cutting, this problem also occurs in scheduling and build formation in additive manufacturing[167,219]. The almost-squares-in-almost-squares (Asqas) problem[271] is also a 2BP|R|F. Search spaces based on signed permutations can be used for this problem by using the IBL encoding. The SOTA for this problem comprises[27,58,116,305,306].

2BP|R|G In this variant of the 2BP with guillotine cuts, the rectangles to pack can be rotated by 90° if need be, but must be arranged such that they can be extracted by edge-to-edge cuts[175,176].

ACO *Ant Colony Optimization* is a nature-inspired optimization method for combinatorial problems where solutions are generated by „ants“ that move from node to node in a graph choosing edges based on (1) the simulated pheromone on the edges and (2) a per-edge heuristic value[83–85]. If an ant produced a good solution, „pheromone“ is distributed over the edges it visited, making it more likely to be re-visited by other ants.

AI Artificial Intelligence, see, e.g., [232]

Glossary (in English) III

Asqas An „almost-square" be a rectangle with size $i \times (i+1)$ with $i \in \mathbb{N}_1$. The *almost-squares-in-almost-squares* problem[271] is a special case of the 2BP|R|F. Here, almost-square rectangles of sizes 1×2, 2×3, $\ldots$, $n \times (n+1)$ must be placed into an almost-square bin without waste. This requires that there exists a natural number $p \in \mathbb{N}_1$ such that $p(p+1) = \sum_{i=1}^{n} i(i+1)$[271]. There are exacty five Asqas instances[271], namely with bin sizes 1×2 for $n = 1$, $p = 1$, 5×4 for $n = 3$, $p = 4$, 15×16 for $n = 8$, $p = 15$, 55×56 for $n = 20$, $p = 55$, and 119×120 for $n = 34$, $p = 119$.

ATSP The general case for TSPs are *asymmetric Traveling Salesperson Problems*, where the $n \times n$ distance matrix A does not need to be symmetric. In other words, it is possible that $\exists a, b \in 1..n$ with $A[a, b] \neq A[b, a]$. Notice that while such index tuple *may* exist, this is not always or necessarily the case. In this case, one may consider the STSP as a special case of the ATSP, while ATSPs may be considered as synonymous to TSPs.

B&B *Branch and bound* is a family of optimization methods that combines systematic exploration of the search space with both and objective function f subject to minimization and a lower bound function. The idea is that we divide the search space into two sets. We then pick the set that seems more interesting and divide it again. We do this recursively until we have arrived at a set of size 1, which is our first solution y. We can compute its objective value $z = f(y)$. Now we continue with the sets that we have created and not yet explored. However, we can discard all sets for which the lower bound of the solution quality is $\geq z$, because they can never give us a better solution than y. If we have divided another set down to size 1 and it yields a better solution than y, of cause we update these values. Eventually, we have either discarded or fully expanded all sets – and then the solution we got must be optimal. B&B-algorithms have been applied to the TSP[172], the JSSP[46], the 2BP|O|F[190], and maritime logistics[298], for instance.

Bash is a shell used under Ubuntu Linux, i.e., the program that „runs" in the terminal and interprets your commands, allowing you to start and interact with other programs[37,205,302]. Learn more at https://www.gnu.org/software/bash.

BP The *Bin Packing Problem*[90,235] is one of the classical problems from Operations Research (OR). The goal is to pack a set of objects into containers, using as few containers as possible. Different flavors of BPs can be distinguished, e.g., by the number of dimensions[73,174] that the objects and containers have (see, e.g., the 2BP), and/or whether the objects can be rotated[306], as well as other constraints and limitations. Both the one- and two-dimensional BP are strongly $\mathcal{NP}$-hard[176].

Glossary (in English) IV

BRKGA The *Biased Random Key Genetic Algorithm* is a EA with a continuous search space $\mathbb{X} \subset [0,1]^n$ and a decoding function γ translating this space to something else and biased, uniform, and elitist mating strategies[118,179].The *bias* comes from the fact that one of the parents used for the binary operation is biased to be of higher fitness than the other parent[118]. Often, γ is the Random Keys encoding[17,18] procedure translating $\mathbb{X}$ to permutations of the first n natural numbers[117]. Sometimes, it is actually a MA further refining the translation result with a local search (LS)[117]. Sometimes the permutations are then further translated to more complex data structures, such as Gantt charts for the JSSP[117], or packing plans for 2BP 3D packing[116].

C is a programming language, which is very successful in system programming situations[77,222,282].

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

client-server architecture is a system design where a central server receives requests from one or multiple clients[26,180,209,225,230]. These requests and responses are usually sent over network connections. A typical example of such a system is the World Wide Web (WWW), where web servers host websites and make them available to web browsers, the clients. Another typical example is the structure of database (DB) software, where a central server, the DBMS, offers access to the DB to the different clients. Here, the client can be some terminal software shipping with the DBMS, such as psql, or the different applications that access the DBs.

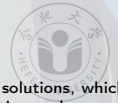
Concorde is an exact solver for the TSP[4,62,63], which could even solve the largest instance of TSPLib (with 85 900 cities) to optimality.

CSP The *Cutting Stock Problem* is about cutting shapes of a larger shape while reducing waste[176,184]. It is closely related to the BP.

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*[279].

DBMS A *database management system* is the software layer located between the user or application and the DB. The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB[300].

Glossary (in English) V



EA An *evolutionary algorithm* is a metaheuristic optimization method that maintains a population of candidate solutions, which undergo selection (where better solutions are chosen with higher probability) and reproduction (where mutation and recombination create a new candidate solution from one or two existing ones, repectively)[10,280].

f-string let you include the results of expressions in strings[39,106,109,120,191,242]. They can contain expressions (in curly braces) like `f"a{6-1}b"` that are then transformed to text via (string) interpolation, which turns the string to `"a5b"`. F-strings are delimited by `f"..."`.

FFA *Frequency Fitness Assignment* (中文: 频率适应度分配) is a algorithm plugin for optimization methods applied to discrete or combinatorial problems with not-too-many different possible objective values[286,288,289]. It replaces the objective values in all comparisons with their absolute encounter frequency so far during the search. FFA has successfully been applied to the Maximum Satisfiability (MaxSAT)[288,289], the TSP[168,169], and the QAP[55,264], amongst others. Its properties have been intensely researched on discrete benchmark functions such as OneMax, LeadingOnes, Trap, TwoMax, Jump, and Plateau[288,289,303]. The most common realizations of FFA so far are the $(1+1)$ FEA and FRLS.

FSSP The *Flow Shop Scheduling Problem* is a variant of a JSSP where all products go through the same sequence of machines[121,147].

GA A *Genetic Algorithm* is an EA usually for bit-string based search spaces[10,70,115,195,201,280] (although many researchers use the terms EA and GA synonymously). It is a population-based metaheuristic applying a unary and a binary search operater as well as a selection scheme. The original GA is the Standard Genetic Algorithm (SGA), which today is considered outdated. Modern variants are the Greedy $(2+1)$ GA and the Self-Adjusting $(1+(\lambda,\lambda))$ Genetic Algorithm (Self-Adjusting $(1+(\lambda,\lambda))$ GA) as well as bit-string based $(\mu+\lambda)$ EAs.

Gantt chart Gantt charts are two-dimensional diagrams illustrating the steps of a process[150,295]. Their horizontal axis represents the time, the vertical axis represents work packages, machines, or workers. In the diagram area, bars located at certain vertical locations indicate that the work package, machine, or worker is processed or works at the time range covered by the bar as projected to the horizontal axis.

GECCO The *Genetic and Evolutionary Computation Conference* is one of the largest conferences in the field of EAs. It has been established in 1999 and its proceedings are published by ACM. Learn more at https://sig.sigevo.org/GECCOs.

Glossary (in English) VI



Git is a distributed Version Control System (VCS) which allows multiple users to work on the same code while preserving the history of the code changes[241,269]. Learn more at https://git-scm.com.

GitHub is a website where software projects can be hosted and managed via the Git VCS[215,269]. Learn more at https://github.com.

GRASP A *Greedy Randomized Adaptive Search Procedure* is a metaheuristic where each cycle consists of two main components[95,97,98]: First, a starting point is generated using a heuristic method (as opposed to being randomly sampled). Then, a LS is applied to refine the solution. Then, another cycle starts and proceeds in the same way. The overall best solution is retained as result. Many examples of applications of GRASPs are listed in [98]. GRASP in combination with VND has been applied to the 2BP|O|F in [213].

Greedy $(2+1)$ GA The Greedy $(2+1)$ GA[47,48,252,253] is an EA for bit-string based search spaces, i.e., a Genetic Algorithm (GA). It was shown to be twice as fast as any mutation-only EA on OneMax[252,253].

IBL The *improved bottom left* heuristic is a one-shot algorithm constructing a solution for 2BP[173]. For the 2BP|O|F problem variant, the input of the IBL algorithm is a permutation of the first n natural numbers, each number identifying one rectangle to be packed. For the 2BP|R|F problem flavor, the permutation elements are signed, where a negative sign indicates a rotation by 90° [173]. In[305,306] it is adapted to a multi-bin situation as follows: The algorithm iterates over the (potentially signed) permutation x and places the rectangles into the packing plan in the prescribed order. It first places the rectangle outside on top of the current bin b, with its bottom-right corner onto the top-right corner of the current bin b. It then moves the rectangle downwards and leftwards as far as possible, prioritizing the downward movement whenever possible. If the rectangle cannot be moved any further, it checks if it is completely contained in the bin b. If yes, it can remain there. Otherwise, a new bin is opened ($b \leftarrow b+1$) and the rectangle is placed at its bottom-left corner. Once a new bin is opened, this bin is used for all further insertions. IBL is suitable as a decoding function γ inside an EA[173]. Historically, it improves upon the BL algorithm[142]: That algorithm cannot represent all possible optimal packings using a permutation, but the IBL can[173].

ILS An *iterated local search* is a variant of the LS which applies cycles of local search and perturbation[251]. First, an initial solution is generated and refined by the local search. Then the cycle begins: A perturbation, i.e., a large search move that likely destroys a good part of the solution, is applied. The resulting new solution is refined by local search. Then, an acceptance criterion decides whether we should use this or the previous solution to continue in the next cycle. ILS has successfully been applied to the QAP.

Glossary (in English) VII



Ising Model The Ising models are benchmark problems for bit-string based black-box search[81,99,100,303]. Here, a neighborhood is defined around each bit. The bit needs to have the same value as all bits in the neighborhood structure T. Thus, we try to achieve that all bits have the same value. The objective function $f(x)$ of the minimization variant of this problem family can be defined as $f(x,T) = \sum_{(i,j)\in T} |x_i - x_j|$, where T contains the index pairs of neighboring bits and is defined differently for the Ising model instances Ising 1D and Ising 2D. The problems are highly epistatic, as all variables interact with some other variables.

Ising 1D The 1D is a benchmark problem for bit-string based black-box search[81,100,303]. The bits are arranged in a one-dimensional ring and we try to minimize the number of bits that have different values than their neighbors. Hence, the neighborhood structure T is defined as $T_{\mathbf{1}D} = \{(i+1, (i+1 \bmod n)+1) \mid i \in [0..n-1]\}$. The problems are highly epistatic, as all variables interact with some other variables. The $(1+1)$ EA has an expected runtime in $\mathcal{O}(n^{\mathbf{3}})$ on 1-Dimensional Ising Model (Ising 1D)[100].

Ising 2D The 2D is a benchmark problem for bit-string based black-box search[81,99,303]. The bits are arranged in a two-dimensional torus and we try to minimize the number of bits that have different values than their neighbors. Hence, the neighborhood structure $T = T_{\mathbf{2}D}$ is defined as the set of all $\{(\alpha+\beta N, \gamma+\delta N)\}$ pairs with $\alpha,\beta,\gamma,\delta \in 0..N-1$ and either ()$\gamma = (\alpha+1) \bmod N$ and $\delta = \beta$) or ($\gamma = \alpha$ and $\delta = (\beta+1) \bmod N$) with $n = N^{\mathbf{2}}$. The problems are highly epistatic, as all variables interact with some other variables. On a similar class of problems, the $(1+1)$ EA has an exponential expected runtime[99].

IT information technology

Java is another very successful programming language, with roots in the C family of languages[31,181].

JSSP The *Job Shop Scheduling Problem*[30,57,121,161] is one of the most prominent and well-studied scheduling tasks. In a JSSP instance, there are m machines and k jobs[254]. Each job must be processed once by each machine in a job-specific sequence and has a job-specific processing time on each machine. The goal is to find an assignment of jobs to machines that results in an overall shortest makespan, i.e., the schedule (Gantt chart) which can complete all the jobs in the shortest time. The JSSP is $\mathcal{NP}$-complete[53,160,161]. The 8 most common sets of JSSP benchmark instances are *abz*[1], *dmu*[74], *ft*[101], *la*[163], *orb*[5], *swv*[249], *ta*[254], and *yn*[299].Their data can be found (sometimes partially) in several repositories, such as the OR-Library[19,20], [238], and [281], the latter of which also contains many results from existing works. Encodings based on permutations with repetition are proposed in [28, 29, 110, 237] and used in [288]. The SOTA of the JSSP includes[117,131,212,214,239,277,304].

Glossary (in English) VIII

Jump The Jump problem is a benchmark problem for bit-string based black-box search[87,105,303]. Like the OneMax, it tries to maximize the number of 1 bits in a bit string of chosen length n. However, the solutions representing the last ω bits around the optimum to flip are deceptive, meaning that they are actually worse and getting worse than surrounding solutions. The objective function $f(x)$ of the minimization variant is $n - \|x\|_1$ if $\|x\|_1 = n$ or $\|x\|_1 \leq n - \omega$ hold, and $\omega + \|x\|_1$ otherwise, where $\|x\|_1 = \sum_{i=1}^{n} x_i$. This means that, the larger ω, the larger the deceptive basin around the optimum that an algorithm needs to cross. An $(1+1)$ EA will converge to the resulting local optimum and thus have an expected runtime of $\Theta(n^\omega + n \ln n)$[87].In experiments, the $(1+1)$ FEA achieves polynomial runtime on Jump, seemingly in $\mathcal{O}(n^2 \ln n)$[288,289,303].

LABS *Low-Autocorrelation Binary Sequences* are bit strings of length n whose autocorrelation is as low as possible[33,34,38,75,76,107,193,196,203,210,221]. This means that sub-sequences of any length $1 \leq k < n$ should ideally never repeat. The objective function $f : \{0,1\}^n \mapsto \mathbb{N}_0$ represents the energy of a bit string x and is subject to minimization. We can define it as $f(x) = \sum_{k=1}^{n-1} \left(\sum_{i=0}^{n-1-k} s_i \cdot s_{i+k}\right)^2$ with $s_j = 2x_j - 1$. Such sequences are needed for communication engineering[34,38,203], cryptography[75,203], as well as modulation pulses in radar and sonar ranging[210]. The LABS is $\mathcal{NP}$-hard[267]. The SOTA on LABS contains[33,38,75,76,203].

LAMP Stack A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP[45,128].

LeadingOnes The LeadingOnes problem is a benchmark problem for bit-string based black-box search[231,293,303]. It tries to maximize the number of *leading* 1 bits in the bit string of a chosen dimension n. The objective function $f(x)$ of the minimization variant of this problem can be defined as $f(x) = n - \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$. Different from OneMax, the contribution to the objective value of the bit at index i thus depends on the state of all the bits *before* this index. The problem is unimodal, i.e., has only a single global optimum and no local optimum. Different from OneMax, it is not separable, i.e., we can optimize each variable separately. The expected runtime of $(1+1)$ EA on LeadingOnes is in $\Theta(n^2)$[87]. The average runtime $(1+1)$ FEA on LeadingOnes seems to be slower by factor proportional to n, i.e., seems to be in $\mathcal{O}(n^3)$[288].

linter A linter is a tool for analyzing program code to identify bugs, problems, vulnerabilities, and inconsistent code styles[148,233]. Ruff is an example of a linter used in the Python world.

Glossary (in English) IX



Linux is the leading open source operating system, i.e., a free alternative to Microsoft Windows[11,127,240,268,274]. We recommend using it for this course, for software development, and for research. Learn more at https://www.linux.org. Its variant Ubuntu is particularly easy to use and install.

literal A literal is a specific concrete value, something that is written down as-is[165,261]. In Python, for example, `"abc"` is a string literal, `5` is an integer literal, and `23.3` is a `float` literal. In contrast, `sin(3)` is not a literal. Also, while `5` is an integer literal, if we create a variable `a = 5` then `a` is not a literal either (it is a variable). Hence, literals are values that the Python interpreter reads directly from the source code and creates as objects in memory. They are not something that is the result from a computation or the result of a variable lookup. Python supports some type hints for literals, including the type `LiteralString` for string literals and the type `Literal[xyz]` for arbitrary literals `xyz`.

LK The *Lin-Kernighan* (LK) algorithm[171] is a SLS method for the TSP. The LK algorithm performs iterative k-opt moves, i.e., in each step, it constructs the two sets X and Y of k edges such that removing the edges in X from the current tour T and then adding the edges from Y to T results in a better tour.The LK algorithm is one of the most classical approaches to the TSP.

LKH The *Lin-Kernighan-Helsgaun* algorithm[63,129,130] is a SLS method for the TSP based on the Lin-Kernighan (LK) algorithm[171]. LKH-1 was proposed in [130] and LKH-2 in [129]. It holds the record in the *World TSP*[64] as of 2025.

LS A *local search* is an algorithm where a single „current" candidate solution is maintained and new solutions are selected from its neighborhood in each step. Most LSes are randomized, i.e., stochastic local search (SLS).

MA A *Memetic Algorithm* is an EA where each new solution is refined by a LS before it enters the population[280]. As LS, we can use, e.g., RLS, but TS[56,102] is maybe a better choise. Some SOTA algorithms in some domains may be classified as MAs, e.g., Lin-Kernighan Helsgaun (LKH) basically uses a population that is refined via a local search. MAs also worked well on the JSSP[56,131] and the TSP[102].

MariaDB An open source relational database management system that has forked off from MySQL[6,7,12,91,187,226]. See https://mariadb.org for more information.

Glossary (in English) X



MaxSAT The goal of satisfiability problems is to find an assignment for n Boolean variables that make a given Boolean formula $F : \{0,1\}^n \mapsto \{0,1\}$ become true. In the *Maximum Satisfiability* problem (MaxSAT)[132], F is given in conjunctive normal form, i.e., the variables appear either directly or negated in m „or“ clauses, which are all combined into one „and.“ The objective function $f(x)$, subject to minimization, computes the number of clauses which are false under the variable setting x. If $f(x) = 0$, then all clauses of F are true, which solves the problem. The MaxSAT problem is $\mathcal{NP}$-complete[61] and exhibits low epistasis but deceptiveness[224], making it hard for local searches. Typical benchmark instances can be found in SATLIB, which contains smaller-scale instances, and the up-to-date MaxSAT Evaluation (MSE), providing SOTA benchmarks, algorithms, and results.

metaheuristic A *metaheuristic* is an optimization algorithm which performs a cycle of trial and error during which candidate solutions are sampled from the solution space $\mathbb{Y}$ and evaluated using an objective function $f : \mathbb{Y} \mapsto \mathbb{R}$[113,280]. The results of the objective function are used to decide which solution is promising. The neighborhoods around promising solutions *or* similarly-structured solutions are then sampled in the next cycle. Prominent metaheuristics are SLSes, SA, TS, and EAs.

Microsoft Windows is a commercial proprietary operating system[36]. It is widely spread, but we recommend using a Linux variant such as Ubuntu for software development and for our course. Learn more at https://www.microsoft.com/windows.

ML Machine Learning, see, e.g., [236]

modulo division is, in Python, done by the operator `%` that computes the remainder of a division. `15 % 6` gives us `3`.

moptipy is the *Metaheuristic Optimization in Python* library[287]. It has been used in several different research works, including[55,168–170,264,297,305,306]. Algorithms implemented by moptipy include EAs, $(1+1)$ EA, RLS, SA, and the $(1+1)$ FEA, amongst others. Learn more at https://thomasweise.github.io/moptipy and https://thomasweise.github.io/moptipyapps.

MSE The *MaxSAT Evaluation* is a regularly updated set of benchmark instances for the MaxSAT[139]. It also includes solver software and benchmark results[275].

Mypy is a static type checking tool for Python[166] that makes use of type hints. Learn more at https://github.com/python/mypy and in [283].

MySQL An open source relational database management system[35,91,227,259,294]. MySQL is famous for its use in the LAMP Stack. See https://www.mysql.com for more information.

Glossary (in English) XI

N-Queens The N-Queens Problem is a benchmark problem for bit-string based black-box search[81,303]. The goal is to arrange N queens on an $N \times N$ chessboard so that none can attack the other. The search space has thus dimension $n = N^2$, with each bit standing for a field and encoding whether or not a queen is placed there. The objective function $f(x)$ of the minimization variant is $N - \left(\sum_{i=1}^{n} x_i\right) - N \sum_{\xi} \max\{0, Q_\xi(x) - 1\}$ where $Q_\xi(x)$ is the number of queens in the horizontal, vertical, or diagonale ξ. The problem is quite epistatic.

OneMax OneMax is the simplest possible benchmark problem for bit-string based black-box search[88,204,303]. The goal is to maximize the number of 1 bits in the bit string of a chosen dimension n. The objective function $f(x)$ of the minimization variant of this problem can be defined as $f(x) = n - \sum_{i=1}^{n} x_i$. The problem is unimodal, i.e., has only a single global optimum and no local optimum. It is fully separable, i.e., we can optimize each variable separately. The expected runtime of $(1+1)$ EA on OneMax is in $\Theta(n \ln n)$[79,204]. The average runtime $(1+1)$ FEA on OneMax seems to be slower by factor proportional to n, i.e., seems to be in $\mathcal{O}(n^2 \ln n)$[288].

OR Operations Research (or Operational Research) is the application of sciences such as mathematics and computer science to the management and organization of systems, organizations, enterprises, factories, or projects. It encompasses the development and application of problem-solving methods and techniques (such as mathematical optimization, simulation, queueing theory and other stochastic models) with the goal to improve decision-making and efficiency[71].

OR-Library is a library of benchmark instances for various OR problems[19,20] located at https://people.brunel.ac.uk/~mastjjb/jeb/info.html. It contains instances of the JSSP, 1D- and 2D-Packing, Knapsack, periodic TSPs, amongst others.

PDF The *Portable Document Format*[103,291] is the format in which provide this book. It is the standard format for the exchange of documents in the internet.

Glossary (in English) XII

Plateau The Plateau problem is a benchmark problem for bit-string based black-box search[3,303]. It is structured exactly like Jump, but instead of being deceptive, the last ω bits to flip around the optimum are neutral. The objective function $f(x)$ of the minimization variant is $n - \|x\|_{\mathbf{1}}$ if $\|x\|_{\mathbf{1}} = n$ or $\|x\|_{\mathbf{1}} \leq n - \omega$ hold, and ω otherwise, where $\|x\|_{\mathbf{1}} = \sum_{i=\mathbf{1}}^{n} x_i$. This means that, the larger ω, the larger the neutral plane around the optimum that an algorithm needs to cross. An $(1+1)$ EA has an expected runtime of $\mathbf{\Theta}(n^{\omega})$ on plateaus[3]. The average runtime of the $(1+1)$ FEA on Plateaus is slower by a factor seemingly proportional in n.

PostgreSQL An open source object-relational DBMS[96,208,220,259]. See https://postgresql.org for more information.

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

Python The Python programming language[135,164,183,283], i.e., what you will learn about in our book[283]. Learn more at https://python.org.

QAP The *Quadratic Assignment Problem* is an optimization problem where the goal is to assign a set of n facilities to a set of n locations[42,55,154,177]. Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed. For each QAP instance, a distance matrix A is given, where $A[a, b]$ specifies the distance from location a to location b, as well as a flow matrix B, where $B[i, j]$ is the amount of material flowing (e.g., daily) from facility i to facility j. The objective function $f(x)$ then rates a permutation x as $f(x) = \sum_{i=\mathbf{1}}^{n} \sum_{j=\mathbf{1}}^{n} A[x_i, x_j] * B[i, j]$. The QAP is $\mathcal{NP}$-complete[80,234] and even among this problem class, is considered as particularly hard for metaheuristics[21,42,177,251]. The QAP has applications such as building layout[66,93,155], keyboard layout[44], circuit design[94], wiring[246], and scheduling[244]. Typical benchmark instances can be found in QAPLIB. SOTA algorithms for the QAP include[21,102,194,200,256,257].

QAPLIB is a library of 136 benchmark instances for the Quadratic Assignment Problem (QAP)[41,43], 28 of which have not yet been solved to optimality[2] (at least by 2023).

Glossary (in English) XIII

Random Keys encoding is a decoding procedure $\gamma : \mathbb{X} \mapsto \mathbb{Y}$ that maps a continuous search space $\mathbb{X} \subset [0,1]^n$ which is a subset of the n-dimensional real numbers to the space $\mathbb{Y}$ of permutations of the first n natural numbers[17,18]. For example, on $n = 5$, the vector $x = (0.49, 0.91, 0.33, 0.75, 0.51)$ is translated to the permutation $y = (3, 1, 5, 4, 2)$, because $x_{\mathbf{3}} = 0.33$ has the smallest value, putting 3 at the beginning, $x_{\mathbf{1}} = 0.49$ the second-smallest, placing 1 second, ..., and $x_{\mathbf{2}} = 0.91$ being the largest value meaning that the last number in y will be 2[18]. The advantage of the encoding is that we can search in $\mathbb{X}$ applying arbitrary mathematical operators, crossover, and mutation procedures, but will always get correct permutations. The disadvantage is that $|\mathbb{X}| \gg |\mathbb{Y}|$ and that there actually is no meaning underpinning moves in $\mathbb{X}$ in relation to their projected result in $\mathbb{Y}$.

relational database A relational DB is a database that organizes data into rows (tuples, records) and columns (attributes), which collectively form tables (relations) where the data points are related to each other[60,124,126,243,258,279,292].

RLS Randomized local search retains the best solution x_c discovered so far during the search and, in each step, it applies a unary search operator to this best-so-far solution x_c and derives a new solution x_n. If the new solution x_n is *better or equally good* when compared with x_c, i.e., not worse, then it replaces it, i.e., is stored as the new x_c. If the search space are bit strings of length n, then RLS uses a unary search operator that flips exactly one bit. This operator is the main difference to $(1 + 1)$ EA.

FRLS The RLS with FFA plugged in [303]. On the discrete bit-string domains, this algorithm has a very specific structure: Its unary search operator flips exactly one bit, distinguishing it from the unary operator in the $(1 + 1)$ FEA, which flips each bit with probability $1/n$. On other domains, such as permutations, we use the term synonymously with $(1 + 1)$ FEA.

RTS *Reactive Tabu Search* is a variant of TS where the length of the tabu list is adapted to the problem and to the current evolution of the search, and an escape strategy for diversifying the search is used when that is not sufficient[13–16]. It has successfully been applied to a wide range of problems, from the QAP[14,15], to Knapsack problems[13,15], the N-K landscape[13] to the training of neural networks[16].

Ruff is a linter and code formatting tool for Python[188,189]. Learn more at `https://docs.astral.sh/ruff` or in [283].

SA *Simulated Annealing* is a SLS that sometimes accepts a worse solution[49,141,149,218,280]. The probability to do so decreases over time and with the difference in objective values, i.e.,is the lower the worse the new solution is. SA performed well on a variety of problems, including the QAP[198,265] and the JSSP[212].

Glossary (in English) XIV



SATLIB is a library of benchmark instances for the Maximum Satisfiability (MaxSAT) available at https://www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html[133,134].

Self-Adjusting $(1 + (\lambda, \lambda))$ GA The Self-Adjusting $(1 + (\lambda, \lambda))$ GA is a theory-inspired EA for bit string-based search spaces[48]. It has linear runtime on the OneMax problem[78].

server In a client-server architecture, the server is a process that fulfills the requests of the clients. It usually waits for incoming communication carring the requests from the clients. For each request, it takes the necessary actions, performs the required computations, and then sends a response with the result of the request. Typical examples of servers are web servers[45] in the internet as well as DBMSes. It is also common to refer to the computer running the server processes as server as well, i.e., to call it the „server computer“[157].

SGA The Standard Genetic Algorithm[10,70,115,195,201,280] was the first EA. It maintains a population of solutions and applies mutation and crossover to generate offspring solutions. It uses fitness proportionate selection to choose which solutions should „survive“ into the next generation, which today is considered a very bad design choice, see, e.g., [293].

SLS *Stochastic local search* is a local search (LS), one of the simplest metaheuristic search paradigms[132]: A single „current“ candidate solution is maintained and new solutions are *stochastically* sampled from its neighborhood, i.e., in a way that involves random numbers. If these solutions are deemed interesting based on their objective value, they replace the „current“. The cycle continues. Basic examples are RLS and $(1 + 1)$ EA. More complex examples that add mechanisms to escape local optima on top of the SLS paradigm are TS and SA.

SOTA State-of-the-Art, i.e., the best-performing methods for a field or problem type

SQL The *Structured Query Language* is basically a programming language for querying and manipulating relational databases[51,67–69,136,192,245,247,248,258]. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference[245].

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

STSP A *symmetric Traveling Salesperson Problem* is a special case of the TSP, where the $n \times n$ distance matrix A is symmetric, i.e., $A[a, b] = A[b, a] \forall a, b \in \mathbf{1}..n$.

Glossary (in English) XV



terminal A terminal is a text-based window where you can enter commands and execute them[11,59]. Knowing what a terminal is and how to use it is very essential in any programming- or system administration-related task. If you want to open a terminal under Microsoft Windows, you can Druck auf ⊞+R, dann Schreiben von `cmd`, dann Druck auf ↵. Under Ubuntu Linux, Ctrl+Alt+T opens a terminal, which then runs a Bash shell inside.

Trap The Trap problem is a benchmark problem for bit-string based black-box search[87,206,303]. It is structured exactly like OneMax, however the worst and the best solution are swapped. This makes the problem highly deceptive, as following the „gradient" in the search space leads the search away from the global optimum. For bit strings of length n, the objective function $f(x)$ of the minimization variant is 0 if $\|x\|_1 = 0$ and $n - \|x\|_1 + 1$ otherwise, where $\|x\|_1 = \sum_{i=1}^{n} x_i$. An $(1+1)$ EA will converge to the resulting local optimum and thus have an expected runtime of $\Theta(n^n)$[87]. In experiments, the $(1+1)$ FEA achieves polynomial runtime on Trap, seemingly in $\mathcal{O}(n^2 \ln n)$[288,289,303].

TS *Tabu Search* is one of the most successful SLSes[111,112,114,143]. In order to avoid returning to previous solutions, it declares certain solutions or search moves as „tabu." For example, if a bit in a discrete problem has been flipped by the search operator, it may be forbidden to flip the bit again for some time, thus preventing the algorithm from getting trapped in a cycle. This rule may be overridden if a new best-so-far solution is discovered that outperforms all previous solutions, in which case it would be accepted (this is the so-called aspiration criterion). TS has performed well on the QAP[14,15,197,200,255,256] and on the JSSP[207,214,217,239,304].

TSP In an instance of the *Traveling Salesperson Problem*, also known as *Traveling Salesman Problem*, a set of n cities or locations as well as the distances between them are given[4,123,162,168,284,290]. The goal is to find the shortest round-trip tour that starts at one city, visits all the other cities one time each, and returns to the origin. The TSP is one of the most well-known $\mathcal{NP}$-hard combinatorial optimization problems[108,123,156]. The TSP has many applications[4], including in transportation, logistics, (circuit) plotting and drilling problems[122], telescope aiming, and genome sequencing. Solutions are often encoded as permutations x of the first n natural numbers, where each city gets a number and the order of numbers in the permutation corresponds to the order of cities in the tour. The distances between the n cities are usually given in a $n \times n$ distance matrix A, where $A[a, b]$ is the distance from city a to city b with $a, b \in 1..n$. Then, the objective function subject to minimization is $f(x) = A[x_n, x_1] + \sum_{i=1}^{n-1} A[x_i, x_{i+1}]$. Typical benchmark instances can be found in TSPLib[228,229], the 8th DIMACS challenge[144–146], and the *World TSP* repository[64].

Glossary (in English) XVI



TSPLib is a library of benchmark instances for the Traveling Salesperson Problem (TSP)[228,229]. Its original URL was available until 2025 and then became defunct. The repository https://github.com/shredderzwj/TSPLIB contains the data. Additionally, you may also check https://github.com/thomasWeise/tspSuite, which contains all the TSPLib instances as well as Java code to load them. The TSPLib contains 110 instances of the symmetric Traveling Salesperson Problems (STSPs) with scales n ranging from 14 to 85 900, but 93 instances have less than 2000 nodes. There also are 19 asymmetric Traveling Salesperson Problems (ATSPs) with scales $n \in 17..443$.

TTP The *Traveling Tournament Problem* is the combinatorial optimization problem of both efficiently and fairly organizing a tournament of n teams that play against each other in a pairwise fashion[92,297]. The efficient part boils down to arranging the games such that the total travel length is short, which is somewhat similar to the classical TSP. Initially, each team is at its home location. On each day, a team needs to travel if its scheduled game is not at its present location. On the last day, each team may need to travel back home unless their last game is a home game. The total travel length sums up the lengths of all travels over all teams. The fair part is represented in several constraints, such as doubleRoundRobin, compactness, maxStreak, and noRepeat. The TTP is $\mathcal{NP}$-hard[276].

TwoMax The TwoMax problem is a benchmark problem for bit-string based black-box search[105,272,303]. It has a global and a local optimum, both of about the same size, at opposite ends of the search space. For bit strings of length n, the objective function $f(x)$ of the minimization variant is 0 if $\|x\|_1 = n$ and $1 + n - \max\left\{\|x\|_1, n - \|x\|_1\right\}$ otherwise, where $\|x\|_1 = \sum_{i=1}^{n} x_i$. An $(1+1)$ EA has a 50/50 chance to converge to the local or global optimum and thus has an expected runtime in $\Theta(n^n)$[104,105]. In experiments, the $(1+1)$ FEA achieves polynomial runtime seemingly in $\mathcal{O}(n^2 \ln n)$ on TwoMax[288,289,303].

type hint are annotations that help programmers and static code analysis tools such as Mypy to better understand what type a variable or function parameter is supposed to be[159,273]. Python is a dynamically typed programming language where you do not need to specify the type of, e.g., a variable. This creates problems for code analysis, both automated as well as manual: For example, it may not always be clear whether a variable or function parameter should be an integer or floating point number. The annotations allow us to explicitly state which type is expected. They are *ignored* during the program execution. They are a basically a piece of documentation.

Glossary (in English) XVII



Ubuntu is a variant of the open source operating system Linux[59,128]. We recommend that you use this operating system to follow this class, for software development, and for research. Learn more at https://ubuntu.com. If you are in China, you can download it from https://mirrors.ustc.edu.cn/ubuntu-releases.

UCS Universal Coded Character Set, see Unicode

Unicode A standard for assigning characters to numbers[137,262,270]. The Unicode standard supports basically all characters from all languages that are currently in use, as well as many special symbols. It is the predominantly used way to represent characters in computers and is regularly updated and improved.

URI A *Uniform Resource Identifier* is an identifier for an abstract or physical resource in the internet[301]. It can be a URL, a name, or both. URIs are supersets of URLs. The connection strings of the PostgreSQL DBMS are examples of Uniform Resource Identifiers (URIs).

URL A *Uniform Resource Locator* identifies a resource in the WWW and a way to obtain it by describing a network access mechanism. The most notable example of URLs is the text you write into web browsers to visit websites[25]. URLs are subsets of URIs.

UTF-8 The *UCS Transformation Format 8* is one standard for encoding Unicode characters into a binary format that can be stored in files[137,301]. It is the world wide web's most commonly used character encoding, where each character is represented by one to four bytes. It is backwards compatible with ASCII.

VCS A *Version Control System* is a software which allows you to manage and preserve the historical development of your program code[269]. A distributed VCS allows multiple users to work on the same code and upload their changes to the server, which then preserves the change history. The most popular distributed VCS is Git.

VND *Variable Neighborhood Descend*; see VNS. VND in combination with GRASP has been applied to the 2BP|O|F in [213].

Glossary (in English) XVIII

VNS *Variable Neighborhood Search* is a metaheuristic framework combining a systematic change of neighborhood with LS[125,202]. A *neighborhood* structure in LS defines which solutions can be reached from a given solution within one search step. Usually, one uses a single unary operator and, hence, has a single neighborhood structure. However, for many problems, we can easily design many operators. For permutations, for example, one operator may swap two elements, another one may reverse a section of the permutation, and yet another one, while yet another one may rotate a subsequence one step to the left or right[284]. In the VNS framework, a multiple neighborhoods are investigated in a systematic fashion. Assume that we have a list of neighborhood structures. Variable Neighborhood Descend (VND) works in a deterministic fashion. It finds the best solution that can be reached in one step from the current solution under the current neighborhood. If this is better than the current solution, we accept it and continue with the first neighborhood. Otherwise, we continue with the next neighborhood in the list. In Basic VNS, instead of picking the best move from the current neighborhood, the first improving move is used. Also, at the beginning of each cycle, a random point from the current neighborhood around the current solution is chosen as starting point.

WWW World Wide Web[24,72]

$i!$ The factorial $a!$ of a natural number $a \in \mathbb{N}_1$ is the product of all positive natural numbers less than or equal to a, i.e., $a! = 1 * 2 * 3 * 4 * \cdots * (a-1) * a$[50,89,182].

$i..j$ 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. It always holds that $i..j \subset \mathbb{Z}$. For example, 5..9 is equivalent to $\{5, 6, 7, 8, 9\}$

A the $n \times n$ distance matrix. $A[a, b]$ indicates the distance between locations a and b with $a, b \in 1..n$. A distance matrix can be symmetric, in which case $A[a, b] = A[b, a]\ \forall a, b \in 1..n$. Distance matrices represent the problem instance data in TSPs and part of the instance data of QAPs.

B the $n \times n$ flow matrix. $B[a, b]$ is the flow of goods between locations a and b with $a, b \in 1..n$. Flow matrices are one half of the problem instance data of QAPs.

Glossary (in English) XIX

$f(y)$ The *objective function* $f : \mathbb{Y} \mapsto \mathbb{R}$ can compute a cost or rating for each candidate solution y in the solution space $\mathbb{Y}$. The smaller this value, the better the solution, i.e., in the context of this work, objective functions are subject to minimization. Objective functions are not necessarily continuous and it is also quite common that they have integer or natural numbers as results, e.g., we often have $f : \mathbb{Y} \mapsto \mathbb{N}_0$ instead of $f : \mathbb{Y} \mapsto \mathbb{R}$.

$\gamma(x)$ The decoding function $\gamma : \mathbb{X} \mapsto \mathbb{Y}$ translates point from the search space $\mathbb{X}$ to one point y in the solution space $\mathbb{Y}$. It is a very common case that $\mathbb{X} = \mathbb{Y}$ and that the decoding function γ is the identity mapping. Examples for encodings that translate from a simple search space $\mathbb{X}$ to a complex solution data structure $\mathbb{Y}$ are the IBL algorithm that translates (potentially signed) permutations to two-dimensional packings on the 2BP, the permutation-based encoding of Gantt charts for the JSSP from[28,29,110,237,285], or the permutation-based encoding of game plans for the Traveling Tournament Problem (TTP) from[297]. Encodings can also translate between types of search spaces. The Random Keys encoding, for example, translates from $[0, 1]^n$ to permutations of the first n natural numbers..

k the number of jobs in an instance of the JSSP.

m the number of machines in an instance of the JSSP.

Glossary (in English) XX



n We usually use the value $n \in \mathbb{N}_0$ to denote the *scale*, i.e., the size of the problem to solve. This notion is very general in computer science and not limited to optimization. For example, if we sort a list or search in a list, the scale n is the length of the list, i.e., the number of elements in it. In a TSP, this may be the number of cities. In a discrete benchmark problem such as OneMax or Trap, this may be the number of bits over which the problem is defined. In a QAP, this may be the number of facilities and locations. In a MaxSAT problem, this may be the number of variables that are input to the Boolean formula that we want to become true. There are also problems where the scale is not defined straightforwardly like this. For example, in a JSSP, we have a number of machines and a number of jobs, which together define the size of the problem. But even then, the scale n relates to the size of the solution space $\mathbb{Y}$, and, hence, also of the search space $\mathbb{X}$, which usually grow exponentially with n. And this is already the main issue: The larger n, the (exponentially) larger the number of possible solutions and thus, the larger the runtime of algorithms. This is known as the „curse of dimensionality“[22,54]. Scales n are also often used in conjunction with the $\mathcal{O}$-notation to denote algorithm complexity. For example, comparison-based sorting a list of n items takes $\Theta(n \log n)$ steps[153] (because they need $\lceil \log_2 n! \rceil$ comparisons in the worst case).

$\mathbb{N}_0$ the set of the natural numbers *including* 0, i.e., 0, 1, 2, 3, and so on. It holds that $\mathbb{N}_0 \subset \mathbb{Z}$ and that $\mathbb{N}_0 = \mathbb{N}_1 \cup \{0\}$.

$\mathbb{N}_1$ the set of the natural numbers *excluding* 0, i.e., 1, 2, 3, 4, and so on. It holds that $\mathbb{N}_1 \subset \mathbb{Z}$ and that $\mathbb{N}_1 = \mathbb{N}_0 \setminus \{0\}$.

$\mathcal{NP}$ is the class of computational problems that can be solved in polynomial time by a non-deterministic machine and can be verified in polynomial time by a deterministic machine (such as a normal computer)[119].

$\mathcal{NP}$-complete A decision problem is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and all problems in $\mathcal{NP}$ are reducible to it in polynomial time[119,223]. A problem is $\mathcal{NP}$-complete if it is $\mathcal{NP}$-hard and if it is in $\mathcal{NP}$.

$\mathcal{NP}$-hard A problem is $\mathcal{NP}$-hard if all problems in $\mathcal{NP}$ are reducible to it in polynomial time[119]. Algorithms that guarantee to find the correct solutions of $\mathcal{NP}$-hard problems[53,61,161] have a worst-case runtime that grows exponentially with the problem scale.

$\Omega(g(x))$ If $f(x) = \Omega(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ and $c \in \mathbb{R}^+$ such that $f(x) \geq c * g(x) \geq 0$ for all $x \geq x_0$[65,151,152]. In other words, $\Omega(g(x))$ describes a lower bound for function growth.

$\mathcal{O}(g(x))$ If $f(x) = \mathcal{O}(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ and $c \in \mathbb{R}^+$ such that $0 \leq f(x) \leq c * g(x)$ for all $x \geq x_0$[9,65,151,152,158]. In other words, $\mathcal{O}(g(x))$ describes an upper bound for function growth.

Glossary (in English) XXI

$\Theta(g(x))$ If $f(x) = \Theta(g(x))$, then $f(x) = \mathcal{O}(g(x))$ and $f(x) = \Omega(g(x))$[151,152], meaning that there exist positive constants $c_1, c_2, x_0 \in \mathbb{R}^+$ such that $0 \leq c_1 g(x) \leq f(x) \leq c_2 g(x)$ for all $x \geq x_0$[65]. In other words, $\Theta(g(x))$ describes an exact order of function growth.

$o(g(x))$ If $f(x) = o(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ such that $0 \leq f(x) \leq c * g(x)$ for all $x \geq x_0$ *all* $c \in \mathbb{R}^+$[65]. Different from $\mathcal{O}(g(x))$, $o(g(x))$ holds for *all* constants $c > 0$, whereas $\mathcal{O}(g(x))$ implies the existence of *some* constant $c > 0$. This basically means that $f(x)$ becomes insignificant compared to $g(x)$ as x_0 grows toward infinity. It is often written that $\lim_{x_0 \to \infty} \frac{f(x)}{g(x)} = 0$. In other words, $o(g(x))$ describes an upper bound for function growth that is not asymptotically tight, i.e., an upper bound that grows faster than $f(x)$.

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

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

$\omega(g(x))$ If $f(x) = \omega(g(x))$, then for every positive constant $c > 0$, there exists a number $x_0 \in \mathbb{R}^+$ such that $0 \leq c * g(x) < f(x)$ for all $x \geq x_0$[65]. In other words, $f(x) = \omega(g(x))$ if and only if $g(x) = o(f(x))$. In other words, ω describes a lower bound that is not asymptotically tight.

$\mathbb{X}$ The search space $\mathbb{X}$ is the space through which an optimization algorithm navigates in order to find good solutions. The search operators of metaheuristics are used to sample points $x \in \mathbb{X}$, for example. On many benchmark problems, the objective functions f directly evaluate the elements $x \in \mathbb{X}$, i.e., are defined as $f : \mathbb{X} \mapsto \mathbb{R}$. However, very often these elements x are not the same data structures that the user is interested in and/or that the objective functions evaluate: It is common that the search space $\mathbb{X}$ is a simple and plain data structure that can be tackled with standard operations or mathematical tools, such as real vectors $\mathbb{R}^n$, permutations, or bit strings, whereas the solution space $\mathbb{Y}$ can be something more complicated. Then, the user only cares about the solutions $y \in \mathbb{Y}$ and not about the search space $\mathbb{X}$. A decoding $\gamma : \mathbb{X} \mapsto \mathbb{Y}$ is employed to translate the $x \in \mathbb{X}$ to $y = \gamma(x)$ with $y \in \mathbb{Y}$.

x a point in the search space $\mathbb{X}$.

Glossary (in English) XXII



$\mathbb{Y}$ the set of candidate solutions of an optimization problem is called the *solution space* $\mathbb{Y}$. The solution space corresponds to the data structure holding the candidate solutions y. These are the elements that are passed to the user after the optimization process is completed. They are also the parameters of the objective function(s) $f : \mathbb{Y} \mapsto \mathbb{R}$.

y a candidate solution $y \in \mathbb{Y}$ is one possible solution to an optimization problem. After the optimization process is completed, one or multiple such solutions are passed to the user.

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