合肥大學
HEFEI UNIVERSITY

Programming with Python

21. Dictionaries bzw. Hash Maps

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

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

应用优化研究所
人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2025-09-05

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

Einleitung

Einleitung

- Dictionaries sind Kontainer, die Schlüssel-Wert-Paare (key-value pairs) speichern.

Einleitung

- Dictionaries sind Kontainer, die Schlüssel-Wert-Paare (key-value pairs) speichern.
- Auf die Werte in einem Dictionary kann über die Schlüssel in eckigen Klammern zugegrifen werden.

Einleitung

- Dictionaries sind Kontainer, die Schlüssel-Wert-Paare (key-value pairs) speichern.
- Auf die Werte in einem Dictionary kann über die Schlüssel in eckigen Klammern zugegrifen werden.
- Die Schlüssel sind immer eindeutig und pro Dictionary einmalig, müssen aber unveränderlich sein.

Einleitung

- Dictionaries sind Kontainer, die Schlüssel-Wert-Paare (key-value pairs) speichern.
- Auf die Werte in einem Dictionary kann über die Schlüssel in eckigen Klammern zugegrifen werden.
- Die Schlüssel sind immer eindeutig und pro Dictionary einmalig, müssen aber unveränderlich sein.

Gute Praxis

Die Schlüssel eines Dictionary müssen immer unveränderlich sein.

Einleitung

- Dictionaries sind Kontainer, die Schlüssel-Wert-Paare (key-value pairs) speichern.
- Auf die Werte in einem Dictionary kann über die Schlüssel in eckigen Klammern zugegrifen werden.
- Die Schlüssel sind immer eindeutig und pro Dictionary einmalig, müssen aber unveränderlich sein.
- Wie Mengen basieren Dictionaries auf Hash-Tabellen[2,6,14].

Gute Praxis

Die Schlüssel eines Dictionary müssen immer unveränderlich sein.

Einleitung

- Dictionaries sind Kontainer, die Schlüssel-Wert-Paare (key-value pairs) speichern.
- Auf die Werte in einem Dictionary kann über die Schlüssel in eckigen Klammern zugegrifen werden.
- Die Schlüssel sind immer eindeutig und pro Dictionary einmalig, müssen aber unveränderlich sein.
- Wie Mengen basieren Dictionaries auf Hash-Tabellen[2,6,14].
- Dictionaries können verändert werden. Wir können Einträge hinzufügen und löschen.

Gute Praxis

Die Schlüssel eines Dictionary müssen immer unveränderlich sein.

Beispiele

Erstellen, Type Hints, Einfache Operationen

- Dictionary-Literale werden als `Schlüssel: Wert`-Paare, die durch Kommas getrennt sind, in geschweiften Klammern definiert.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- Dictionary-Literale werden als `Schlüssel: Wert`-Paare, die durch Kommas getrennt sind, in geschweiften Klammern definiert.
- Der Type-Hint `dict[K, V]` definiert, dass ein Dictionary Schlüssel vom Typ `K` und Werte vom Typ `V` hat.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- Dictionary-Literale werden als `Schlüssel: Wert`-Paare, die durch Kommas getrennt sind, in geschweiften Klammern definiert.
- Der Type-Hint `dict[K, V]` definiert, dass ein Dictionary Schlüssel vom Typ `K` und Werte vom Typ `V` hat.
- `len(d)` liefert die Anzahl der Elemente im Dictionary `d`.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- Dictionary-Literale werden als `Schlüssel: Wert`-Paare, die durch Kommas getrennt sind, in geschweiften Klammern definiert.
- Der Type-Hint `dict[K, V]` definiert, dass ein Dictionary Schlüssel vom Typ `K` und Werte vom Typ `V` hat.
- `len(d)` liefert die Anzahl der Elemente im Dictionary `d`.
- `d[k]` liefert den Wert, der unter Schlüssel `k` in Dictionary `d` gespeichert ist.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- Dictionary-Literale werden als `Schlüssel: Wert`-Paare, die durch Kommas getrennt sind, in geschweiften Klammern definiert.
- Der Type-Hint `dict[K, V]` definiert, dass ein Dictionary Schlüssel vom Typ `K` und Werte vom Typ `V` hat.
- `len(d)` liefert die Anzahl der Elemente im Dictionary `d`.
- `d[k]` liefert den Wert, der unter Schlüssel `k` in Dictionary `d` gespeichert ist.
- `d.keys()` liefert eine Kollektion mit den *Schlüseln* vom Dictionary `d`.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- Der Type-Hint `dict[K, V]` definiert, dass ein Dictionary Schlüssel vom Typ `K` und Werte vom Typ `V` hat.
- `len(d)` liefert die Anzahl der Elemente im Dictionary `d`.
- `d[k]` liefert den Wert, der unter Schlüssel `k` in Dictionary `d` gespeichert ist.
- `d.keys()` liefert eine Kollektion mit den *Schlüseln* vom Dictionary `d`.
- Durch die Funktion `list(...)` können wir eine Liste mit den Elementen dieser Kollektion erstellen.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- `len(d)` liefert die Anzahl der Elemente im Dictionary `d`.
- `d[k]` liefert den Wert, der unter Schlüssel `k` in Dictionary `d` gespeichert ist.
- `d.keys()` liefert eine Kollektion mit den *Schlüseln* vom Dictionary `d`.
- Durch die Funktion `list(...)` können wir eine Liste mit den Elementen dieser Kollektion erstellen.
- `d.values()` liefert eine Kollektion mit den *Werten* vom Dictionary `d`.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- `d[k]` liefert den Wert, der unter Schlüssel `k` in Dictionary `d` gespeichert ist.
- `d.keys()` liefert eine Kollektion mit den *Schlüseln* vom Dictionary `d`.
- Durch die Funktion `list(...)` können wir eine Liste mit den Elementen dieser Kollektion erstellen.
- `d.values()` liefert eine Kollektion mit den *Werten* vom Dictionary `d`.
- `d.items()` liefert eine Kollektion mit den Tupeln *(Schlüssel, Wert)* für alle Einträge im Dictionary `d`.

```
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- `d.keys()` liefert eine Kollektion mit den *Schlüseln* vom Dictionary `d`.
- Durch die Funktion `list(...)` können wir eine Liste mit den Elementen dieser Kollektion erstellen.
- `d.values()` liefert eine Kollektion mit den *Werten* vom Dictionary `d`.
- `d.items()` liefert eine Kollektion mit den Tupeln *(Schlüssel, Wert)* für alle Einträge im Dictionary `d`.
- Wir können dem Schlüssel `k` den Wert `w` im Dictionary `d` über `d[k] = w` zuweisen.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- Durch die Funktion `list(...)` können wir eine Liste mit den Elementen dieser Kollektion erstellen.
- `d.values()` liefert eine Kollektion mit den *Werten* vom Dictionary `d`.
- `d.items()` liefert eine Kollektion mit den Tupeln *(Schlüssel, Wert)* für alle Einträge im Dictionary `d`.
- Wir können dem Schlüssel `k` den Wert `w` im Dictionary `d` über `d[k] = w` zuweisen.
- `del d[k]` löscht das Schlüssel-Wert-Paar mit Schlüssel `k` aus Dictionary `d`.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- `d.items()` liefert eine Kollektion mit den Tupeln *(Schlüssel, Wert)* für alle Einträge im Dictionary `d`.
- Wir können dem Schlüssel `k` den Wert `w` im Dictionary `d` über `d[k] = w` zuweisen.
- `del d[k]` löscht das Schlüssel-Wert-Paar mit Schlüssel `k` aus Dictionary `d`.
- `d.pop(k)` löscht das Schlüssel-Wert-Paar mit Schlüssel `k` aus Dictionary `d`, liefert aber den Wert zurück, der vorher unter `k` gespeichert war.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- Wir können dem Schlüssel `k` den Wert `w` im Dictionary `d` über `d[k] = w` zuweisen.
- `del d[k]` löscht das Schlüssel-Wert-Paar mit Schlüssel `k` aus Dictionary `d`.
- `d.pop(k)` löscht das Schlüssel-Wert-Paar mit Schlüssel `k` aus Dictionary `d`, liefert aber den Wert zurück, der vorher unter `k` gespeichert war.
- `d1.update(d2)` fügt alle Schlüssel-Wert-Paare in dem Dictionary `d2` dem Dictionary `d1` hinzu.

```
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

Erstellen, Type Hints, Einfache Operationen

- Wir können dem Schlüssel `k` den Wert `w` im Dictionary `d` über `d[k] = w` zuweisen.
- `del d[k]` löscht das Schlüssel-Wert-Paar mit Schlüssel `k` aus Dictionary `d`.
- `d.pop(k)` löscht das Schlüssel-Wert-Paar mit Schlüssel `k` aus Dictionary `d`, liefert aber den Wert zurück, der vorher unter `k` gespeichert war.
- `d1.update(d2)` fügt alle Schlüssel-Wert-Paare in dem Dictionary `d2` dem Dictionary `d1` hinzu.

```python
"""An example of dictionaries."""

num_str: dict[int, str] = {  # create and type hint a dictionary
    2: "two", 1: "one", 3: "three", 4: "four"}  # the elements
print(f"num_str has {len(num_str)} elements: {num_str}")  # print dict
print(f"I got {num_str[2]} shoes and {num_str[1]} hat.")  # get element

print(f"the keys are: {list(num_str.keys())}")  # print the keys
print(f"the values are: {list(num_str.values())}")  # print the values
print(f"the items are: {list(num_str.items())}")  # the key-value pairs

num_str[5] = "fivv"  # insert (or update) the value of a key
print(f"after adding key 1 num_str is now {num_str}")
num_str[5] = "five"  # update the value of a key
print(f"after updating key 1 num_str is now {num_str}")

del num_str[4]  # delete a key
print(f"after deleting key 4 num_str is now {num_str}")
# get the value of a key, then delete it
print(f"popping key 5 gets us {num_str.pop(5)}")

str_num: dict[str, int] = {}  # create empty dictionary
str_num.update({"one": 1, "three": 3, "two": 2, "four": 4})
print(f"{num_str[1]} + {num_str[2]} = {str_num['three']}")
```

↓ python3 dicts_1.py ↓

```
num_str has 4 elements: {2: 'two', 1: 'one', 3: 'three', 4: 'four'}
I got two shoes and one hat.
the keys are: [2, 1, 3, 4]
the values are: ['two', 'one', 'three', 'four']
the items are: [(2, 'two'), (1, 'one'), (3, 'three'), (4, 'four')]
after adding key 1 num_str is now {2: 'two', 1: 'one', 3: 'three', 4: '
    ↪ four', 5: 'fivv'}
after updating key 1 num_str is now {2: 'two', 1: 'one', 3: 'three', 4:
    ↪ 'four', 5: 'five'}
after deleting key 4 num_str is now {2: 'two', 1: 'one', 3: 'three', 5:
    ↪ 'five'}
popping key 5 gets us five
one + two = 3
```

Zusammenfassung

Zusammenfassung

- Mit Dictionaries kennen wir nun auch der letzte der vier wichtigen Datenstrukturen von Python.

Zusammenfassung

- Mit Dictionaries kennen wir nun auch der letzte der vier wichtigen Datenstrukturen von Python.
- Listen sind veränderliche Sequenzen von Instanzen eines beliebigen Datentyps.

Zusammenfassung

- Mit Dictionaries kennen wir nun auch der letzte der vier wichtigen Datenstrukturen von Python.
- Listen sind veränderliche Sequenzen von Instanzen eines beliebigen Datentyps.
- Tuples sind unveränderliche Sequenzen von Instanzen beliebiger Datentypen (welche selbst unveränderlich seien sollten).

Zusammenfassung

- Mit Dictionaries kennen wir nun auch der letzte der vier wichtigen Datenstrukturen von Python.
- Listen sind veränderliche Sequenzen von Instanzen eines beliebigen Datentyps.
- Tuples sind unveränderliche Sequenzen von Instanzen beliebiger Datentypen (welche selbst unveränderlich seien sollten).
- Mengen beinhalten Instanzen eines unveränderlichen Datentyps. Mengen sind selbst veränderlich und ohne Elementreihenfolge.

Zusammenfassung

- Mit Dictionaries kennen wir nun auch der letzte der vier wichtigen Datenstrukturen von Python.
- Listen sind veränderliche Sequenzen von Instanzen eines beliebigen Datentyps.
- Tuples sind unveränderliche Sequenzen von Instanzen beliebiger Datentypen (welche selbst unveränderlich seien sollten).
- Mengen beinhalten Instanzen eines unveränderlichen Datentyps. Mengen sind selbst veränderlich und ohne Elementreihenfolge.
- Dictionaries sind Zuordnungen von Instanzen eines unveränderlichen Schlüsseldatentyps zu Instanzen eines (möglicherweise veränderlichen) Wertedatentyps. Dictionaries sind veränderlich.

Zusammenfassung

- Mit Dictionaries kennen wir nun auch der letzte der vier wichtigen Datenstrukturen von Python.
- Listen sind veränderliche Sequenzen von Instanzen eines beliebigen Datentyps.
- Tuples sind unveränderliche Sequenzen von Instanzen beliebiger Datentypen (welche selbst unveränderlich seien sollten).
- Mengen beinhalten Instanzen eines unveränderlichen Datentyps. Mengen sind selbst veränderlich und ohne Elementreihenfolge.
- Dictionaries sind Zuordnungen von Instanzen eines unveränderlichen Schlüsseldatentyps zu Instanzen eines (möglicherweise veränderlichen) Wertedatentyps. Dictionaries sind veränderlich.
- In Listen und Tupels kann ein Objekt beliebig oft vorkommen.

Zusammenfassung

- Mit Dictionaries kennen wir nun auch der letzte der vier wichtigen Datenstrukturen von Python.
- Listen sind veränderliche Sequenzen von Instanzen eines beliebigen Datentyps.
- Tuples sind unveränderliche Sequenzen von Instanzen beliebiger Datentypen (welche selbst unveränderlich seien sollten).
- Mengen beinhalten Instanzen eines unveränderlichen Datentyps. Mengen sind selbst veränderlich und ohne Elementreihenfolge.
- Dictionaries sind Zuordnungen von Instanzen eines unveränderlichen Schlüsseldatentyps zu Instanzen eines (möglicherweise veränderlichen) Wertedatentyps. Dictionaries sind veränderlich.
- In Listen und Tupels kann ein Objekt beliebig oft vorkommen.
- Jedes Element einer Menge und jeder Schlüssel eines Dictionaries kann immer nur einmal vorkommen.

Zusammenfassung

- Mit Dictionaries kennen wir nun auch der letzte der vier wichtigen Datenstrukturen von Python.
- Listen sind veränderliche Sequenzen von Instanzen eines beliebigen Datentyps.
- Tuples sind unveränderliche Sequenzen von Instanzen beliebiger Datentypen (welche selbst unveränderlich seien sollten).
- Mengen beinhalten Instanzen eines unveränderlichen Datentyps. Mengen sind selbst veränderlich und ohne Elementreihenfolge.
- Dictionaries sind Zuordnungen von Instanzen eines unveränderlichen Schlüsseldatentyps zu Instanzen eines (möglicherweise veränderlichen) Wertedatentyps. Dictionaries sind veränderlich.
- In Listen und Tupels kann ein Objekt beliebig oft vorkommen.
- Jedes Element einer Menge und jeder Schlüssel eines Dictionaries kann immer nur einmal vorkommen.
- Tupels und Listen sind besonders geeignet, wenn wir Elemente in einer bestimmten Reihenfolge speichern und auslesen wollen. Wenn wir die Sequenz verändern wollen, dann nehmen wir Listen, sonst Tuples.

Zusammenfassung

- Listen sind veränderliche Sequenzen von Instanzen eines beliebigen Datentyps.
- Tuples sind unveränderliche Sequenzen von Instanzen beliebiger Datentypen (welche selbst unveränderlich seien sollten).
- Mengen beinhalten Instanzen eines unveränderlichen Datentyps. Mengen sind selbst veränderlich und ohne Elementreihenfolge.
- Dictionaries sind Zuordnungen von Instanzen eines unveränderlichen Schlüsseldatentyps zu Instanzen eines (möglicherweise veränderlichen) Wertedatentyps. Dictionaries sind veränderlich.
- In Listen und Tupels kann ein Objekt beliebig oft vorkommen.
- Jedes Element einer Menge und jeder Schlüssel eines Dictionaries kann immer nur einmal vorkommen.
- Tupels und Listen sind besonders geeignet, wenn wir Elemente in einer bestimmten Reihenfolge speichern und auslesen wollen. Wenn wir die Sequenz verändern wollen, dann nehmen wir Listen, sonst Tuples.
- Mengen (und Dictionaries) sind dann besonders nützlich, wenn wir schnell nachschlagen wollen, ob ein bestimmtes Element irgendwo in der Kollektion (als Schlüssel) vorkommt.

Überblick

Type Hint	`list[A]`	`tuple[B, C, ...]`	`set[D]`	`dict[E, F]`
Literal	`[a1, a2, ...]`	`(b, c, ...)`	`{d1, d2, ...}`	`{e1: f1, e2: f2, ...}`
leeres Literal	`[]`	`()`	✗ / `set()`	`{}`
andere Kollektion kopieren `x`	`list(x)`	`tuple(x)`	`set(x)`	`dict(x)` (`x` is `dict`)
Geordnet	✓	✓	✗	✓ (insertion sequence)
Kollektion veränderlich	✓	✗	✓	✓
veränderliche Werte	✓	✗	✗	`E:` ✗; `F:` ✓
Alle Elemente selber Datentyp	✓	✗	✓	✓
Indizieren über `[i]`	✓ (`i` is `int`)	✓ (`i` is `int`)	✗	✓ (`i` is `E`)
Element mehrfach	✓	✓	✗	`E:` ✗; `F:` ✓
Element hinzufügen	$\mathcal{O}(1)$	✗	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`in`/`not in`	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Element löschen	$\mathcal{O}(n)$	✗	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Overhead	sehr klein	sehr klein	ja	ja

谢谢您们！

Thank you!

Vielen Dank!

References I

[1] Paul Gustav Heinrich Bachmann. *Die Analytische Zahlentheorie / Dargestellt von Paul Bachmann*. Bd. Zweiter Theil der Reihe Zahlentheorie: Versuch einer Gesamtdarstellung dieser Wissenschaft in ihren Haupttheilen. Leipzig, Sachsen, Germany: B. G. Teubner, 1894. ISBN: 978-1-4181-6963-3. URL: http://gallica.bnf.fr/ark:/12148/bpt6k994750 (besucht am 2023-12-13) (siehe S. 40).

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald Linn Rivest und Clifford Stein. *Introduction to Algorithms*. 3. Aufl. Cambridge, MA, USA: MIT Press, 2009. ISBN: 978-0-262-03384-8 (siehe S. 5–10).

[3] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 39).

[4] 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. 40).

[5] 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. 40).

[6] 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. 5–10).

[7] 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. 40).

[8] 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. 39).

[9] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 39).

[10] 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. 39).

References II

[11] 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. 39).

[12] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 39).

[13] 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. 39).

[14] Abraham "Avi" Silberschatz, Henry F. "Hank" Korth und S. Sudarshan. *Database System Concepts*. 7. Aufl. New York, NY, USA: McGraw-Hill, März 2019. ISBN: 978-0-07-802215-9 (siehe S. 5–10).

[15] Anna Skoulikari. *Learning Git*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mai 2023. ISBN: 978-1-0981-3391-7 (siehe S. 39).

[16] ."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. 39).

[17] 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. 39).

[18] 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. 39).

[19] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (besucht am 2025-01-05) (siehe S. 39).

Glossary (in English) I

Git is a distributed Version Control Systems (VCS) which allows multiple users to work on the same code while preserving the history of the code changes[15,17]. Learn more at https://git-scm.com.

GitHub is a website where software projects can be hosted and managed via the Git VCS[13,17]. Learn more at https://github.com.

literal A literal is a specific concrete value, something that is written down as-is[10,16]. 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`.

Mypy is a static type checking tool for Python[11] that makes use of type hints. Learn more at https://github.com/python/mypy and in[19].

Python The Python programming language[3,9,12,19], i.e., what you will learn about in our book[19]. Learn more at https://python.org.

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[8,18]. 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.

VCS A *Version Control System* is a software which allows you to manage and preserve the historical development of your program code[17]. 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.

Glossary (in English) II

$\Omega(g(x))$ If $f(x) = \Omega(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ and $c \in \mathbb{R}^+$ such that $f(x) \geq c * g(x) \geq 0 \forall x \geq x_0$[4,5]. In other words, $\Omega(g(x))$ describes a lower bound for function growth.

$\mathcal{O}(g(x))$ If $f(x) = \mathcal{O}(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ and $c \in \mathbb{R}^+$ such that $0 \leq f(x) \leq c * g(x) \forall x \geq x_0$[1,4,5,7]. In other words, $\mathcal{O}(g(x))$ describes an upper bound for function growth.

$\Theta(g(x))$ If $f(x) = \Theta(g(x))$, then $f(x) = \mathcal{O}(g(x))$ and $f(x) = \Omega(g(x))$[4,5]. In other words, $\Theta(g(x))$ describes an exact order of function growth.

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

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