



PCEP-30-02 - Tuples, Dictionaries und Strings (3.2, 3.3, 3.4)

PCEP-30-02 – Tuples, Dictionaries und Strings (3.2, 3.3, 3.4)

Lernziele – Überblick über Tuples, Dictionaries und Strings

- **Tuples:** Erstellung, Zugriff, Unterschiede zu Listen
- **Dictionaries:** Schlüssel-Wert-Paare, Methoden, Iteration
- **Strings:** Textverarbeitung, Methoden, Formatierung
- **Gemeinsamkeiten und Unterschiede dieser Datentypen**

Übersicht der Themen – Struktur des Slide-Decks

- **Teil 1:** Tuples – Daten sammeln und verarbeiten
- **Teil 2:** Dictionaries – Schlüssel-Wert-Paare
- **Teil 3:** Strings – Zeichenketten und Textverarbeitung
- **Multiple-Choice-Fragen und Challenges**

Einführung in Tuples: Definition, Syntax, Beispiel

- **Definition:** Tuples sind **unveränderliche**, geordnete Datensammlungen
- **Syntax:**

```
fruits = ("Apfel", "Banane", "Kirsche")
print(fruits)
```

Erstellung und Initialisierung von Tuples

- Einfache Tuples:

```
numbers = (1, 2, 3)
```

- Einzelnes Element (Komma erforderlich):

```
single = (42,)
```

Indexing und Slicing von Tuples

- Beispiel:

```
t = (10, 20, 30, 40)
print(t[0])    # 10
print(t[-1])   # 40
print(t[1:3])  # (20, 30)
```

Blitzfrage (Tuples): Verhalten bei Zuweisung in Tuples

Was passiert bei folgendem Code?

```
t = (1, 2, 3)  
t[1] = 5
```

- A) 5 wird eingesetzt
- B) Es entsteht ein Fehler
- C) Das Tupel wird automatisch eine Liste
- D) Das Programm stürzt ab

Blitzantwort (Tuples): TypeError – Unveränderlichkeit

Richtige Antwort: B) Es entsteht ein Fehler
(TypeError)

Grund: Tuples sind unveränderlich (immutable).

Unterschiede zwischen Tupeln und Listen (Tabelle)

Merkmal	Tupel	Liste
Veränderlich?	 Nein	 Ja
Performance	 Schneller	 Langsamer
Speicherplatz	 Effizienter	 Mehr Platz
Verschachtelbar	 Ja	 Ja

Häufige Fehler bei Tuples (z. B. Einzelnelement-Tuples)

- Fehler:

```
single = (42)    # Kein Tupel, sondern Zahl
```

- Korrekt:

```
single = (42,)    # Einzel-Element-Tuple
```

Best Practices bei der Verwendung von Tuples

-  Nutze Tuples für feste, unveränderliche Daten
-  Bevorzuge Tuples gegenüber Listen, wenn keine Änderungen erforderlich sind
-  Keine verschachtelten Tuples ohne klare Struktur

Tuple-Methoden: `count()`, `index()`

- Anzahl der Vorkommen zählen (`count`):

```
numbers = (1, 2, 2, 3)
print(numbers.count(2)) # 2
```

- Index eines Elements finden (`index`):

```
fruits = ("Apfel", "Banane", "Kirsche")
print(fruits.index("Banane")) # 1
```

Verschachtelte Tuples und Tuple Unpacking

- Verschachtelte Tuples:

```
nested = ((1, 2), (3, 4))  
print(nested[1][0]) # 3
```

- Tuple Unpacking:

```
coordinates = (52.5200, 13.4050)  
lat, lon = coordinates  
print(lat, lon)
```

Challenge 1: Tuple Unpacking (Längen- und Breitenkoordinaten)

Aufgabe:

- Entpacke ein Koordinaten-Tuple
- Gib Latitude und Longitude aus

```
coordinates = (52.5200, 13.4050)
lat, lon = coordinates
print(f"Latitude: {lat}, Longitude: {lon}")
```

Challenge 2: Zugriff auf Elemente in verschachtelten Tuples

Aufgabe:

- Greife auf die innere Zahl 42 zu

```
nested = ((10, 20), (30, (40, 42)))
print(nested[1][1][1])  # 42
```

Multiple-Choice Frage 1 (Tuples):

.count() Methode

Welcher Wert wird ausgegeben?

```
numbers = (1, 2, 2, 3)
print(numbers.count(2))
```

- A) 1
- B) 2
- C) 3
- D) Fehler

Richtige Antwort: B) 2

Multiple-Choice Frage 2 (Tuples): Einzel-Element-Tuple

Was ist die korrekte Definition eines Einzel-Element-Tuples?

- A) single = (42)
- B) single = (42,)
- C) single = [42]
- D) single = \{42\}

Richtige Antwort: B) `single = (42,)` –
Einzelnelement-Tuple erfordert ein Komma.

Multiple-Choice Frage 3 (Tuples): Vergleich von Tuples und Listen

Eigenschaft	Tuple	Liste
Veränderlich	<input checked="" type="checkbox"/> Nein	<input checked="" type="checkbox"/> Ja
Geordnet?	<input checked="" type="checkbox"/> Ja	<input checked="" type="checkbox"/> Ja
Schneller	<input checked="" type="checkbox"/> Ja	<input checked="" type="checkbox"/> Nein

Multiple-Choice Frage 4 (Tuples): Zugriffsarten (Slicing, Indexing)

Was gibt dieser Code aus?

```
t = (5, 10, 15, 20)  
print(t[1:3])
```

- A) (10, 15)
- B) (5, 10)
- C) (15, 20)
- D) Fehler

Richtige Antwort: A) (10, 15) – Slicing schneidet von Index 1 bis 2.

Multiple-Choice Frage 5 (Tuples): Erstellen von leeren Tuples

Welcher Befehl erzeugt ein leeres Tuple?

- A) tuple = ()
- B) tuple = []
- C) tuple = {}
- D) tuple = tuple

Richtige Antwort: A) tuple = ()

Einführung in Dictionaries: Definition, Syntax, Beispiel

- **Definition:** Ein Dictionary ist eine **GEORDNETE** Sammlung von Schlüssel-Wert-Paaren. (ab Python 3.7, Danke Jürgen)
- **Syntax:**

```
person = {  
    "name": "Alice",  
    "age": 30,  
    "city": "Berlin"  
}  
  
print(person["name"]) # Alice
```


Erstellen und Initialisieren von Dictionaries

- Leeres Dictionary:

```
d = {}
```

- Mit Schlüssel-Wert-Paaren:

```
d = {"name": "Alice", "age": 30}
```

Zugriff auf Werte in Dictionaries (dict[key], .get())

```
person = {"name": "Bob", "age": 25}
print(person["name"])    # Bob
print(person.get("age")) # 25
print(person.get("city", "Unbekannt")) # Unbekannt
```

Blitzfrage (Dictionaries): Zugriff auf einen nicht vorhandenen Schlüssel

Was gibt dieser Code aus?

```
person = {"name": "Alice"}  
print(person.get("age"))
```

- A) None
- B) 0
- C) Fehler
- D) "Unbekannt"

Richtige Antwort: A) None – .get() gibt None zurück, wenn der Schlüssel fehlt.

Blitzantwort (Dictionaries):

.get() gibt None zurück

Richtige Antwort: A) None

- .get() ist sicherer als person["age"], da es keinen Fehler wirft, wenn der Schlüssel fehlt.

Wichtige Dictionary-Methoden: .keys(), .values(), .items()

```
person = {"name": "Alice", "age": 30}

print(person.keys())    # dict_keys(['name', 'age'])
print(person.values()) # dict_values(['Alice', 30])
print(person.items())  # dict_items([('name', 'Alice'), ('age', 30)])
```

Überprüfung der Existenz von Schlüsseln (in Operator)

```
person = {"name": "Alice"}  
print("name" in person)  # True  
print("age" in person)  # False
```

Hinzufügen und Entfernen von Schlüsseln (update(), pop(), del)

```
d = {"name": "Alice"}  
  
# Hinzufügen  
d["age"] = 30  
d.update({"city": "Berlin"})  
  
# Entfernen  
d.pop("age")  
del d["name"]  
  
print(d) # {'city': 'Berlin'}
```

Iteration durch Dictionaries (for key, value in dict.items())

```
person = {"name": "Alice", "age": 30}

for key, value in person.items():
    print(f"{key}: {value}")
```

Challenge 1: Telefonbuch-Dictionary erstellen und ausgeben

Aufgabe:

- Erstelle ein Dictionary für ein **Telefonbuch**
- Füge mindestens 2 Einträge hinzu
- Gib alle Namen und Nummern aus

```
phonebook = {"Alice": 12345, "Bob": 67890}

for name, number in phonebook.items():
    print(f"{name}: {number}")
```

Challenge 2: Schlüssel-Wert-Paare filtern (Alter > 30)

Aufgabe:

- Erstelle ein Dictionary mit Namen und Alter
- Filtere Personen, die älter als 30 sind

```
people = {"Alice": 25, "Bob": 35, "Charlie": 40}
for name, age in people.items():
    if age > 30:
        print(f"{name} ist {age} Jahre alt.")
```

Häufige Fehler bei Dictionaries (**KeyError**, doppelte Schlüssel)

- **KeyError:** Zugriff auf nicht vorhandene Schlüssel

```
person = {"name": "Alice"}  
print(person["age"]) # KeyError
```

- **Doppelte Schlüssel:** Der letzte Wert überschreibt vorherige

```
d = {"a": 1, "a": 2}  
print(d) # {'a': 2}
```

Best Practices für effiziente Dictionary-Nutzung

-  Nutze `.get()` für sicheren Zugriff
-  Verwende `.setdefault()` zur Initialisierung
-  Nutze Dictionary-Comprehensions für effiziente Filter
-  Vermeide doppelten Schlüssel-Einsatz

Multiple-Choice Frage 1

(Dictionaries): Existenzprüfung von Schlüsseln

Wie prüfst du, ob ein Schlüssel existiert?

- A) if key.has(dict):
- B) if key in dict:
- C) if dict.key():
- D) if dict.hasKey(key):

Richtige Antwort: B) if key in dict:

Multiple-Choice Frage 2

(Dictionaries): Unterschied zwischen .get() und []

Was ist der Unterschied zwischen .get() und []?

- A) .get() wirft immer einen Fehler
- B) .get() gibt None zurück, wenn der Schlüssel fehlt
- C) [] gibt None zurück
- D) Sie sind identisch

Richtige Antwort: B) .get() gibt None zurück,
wenn der Schlüssel fehlt

Multiple-Choice Frage 3

(Dictionaries): Iteration über .items()

Welche Schleife iteriert korrekt über Schlüssel-Wert-Paare?

- A) for x in dict:
- B) for key, value in dict:
- C) for key, value in dict.items():
- D) for item in dict.values():

Richtige Antwort: C) for key, value in
dict.items():

Multiple-Choice Frage 4

(Dictionaries): Unterschied zwischen `.update()` und `.setdefault()`

Was macht `.setdefault()` im Vergleich zu `.update()`?

- A) Fügt immer neue Schlüssel hinzu
- B) Überschreibt vorhandene Schlüssel
- C) Fügt nur hinzu, wenn der Schlüssel nicht

existiert

D) Entfernt vorhandene Schlüssel

Richtige Antwort: C) `.setdefault()` fügt nur hinzu, wenn der Schlüssel nicht existiert

Multiple-Choice Frage 5 (Dictionaries): Verwendung von immutablen Schlüsseln

Welcher der folgenden Typen kann **kein** Schlüssel in einem Dictionary sein?

- A) str
- B) int
- C) tuple
- D) list

Richtige Antwort: D) list – Listen sind
veränderlich und daher ungeeignet als Schlüssel

Blitzfrage (Dictionaries): Welcher Schlüssel ist valide?

Welche der folgenden Definitionen ist korrekt?

- A) `d = {["a"] : 1}`
- B) `d = {(1, 2) : "value"}`
- C) `d = {{1: 2} : 3}`
- D) `d = {None: "value"}`

Richtige Antwort: B) `d = {(1, 2): "value"}`

– Tuples sind immutable und gültig als Schlüssel

Blitzantwort (Dictionaries): Nur immutable Typen sind gültig

-  **Schlüssel müssen unveränderlich (immutable) sein:**
-  **Listen und Sets sind nicht erlaubt:** Sie sind veränderlich

Einführung in Strings: Definition, Syntax, Beispiel

- **Definition:** Eine Zeichenkette (String) ist eine Folge von Zeichen.
- **Beispiel:**

```
text = "Hallo, Welt!"  
print(text)
```

Erstellung von Strings (einfache, mehrzeilige, Rohstrings)

- Einfache Strings: "Hallo" oder 'Hallo'
- Mehrzeilige Strings:

```
text = """Dies ist  
ein mehrzeiliger  
String."""
```

String-Methoden (.strip(), .lower(), .upper(), .replace())

```
text = " Python "
print(text.strip())      # "Python"
print(text.lower())      # " python "
print(text.upper())      # " PYTHON "
print(text.replace("Python", "Java")) # " Java "
```

Blitzfrage (Strings): Ergebnis von `s[::-1]` (String umdrehen)

Was gibt dieser Code aus?

```
s = "Python"  
print(s[::-1])
```

- A) "Python"
- B) "nohtyP"
- C) "thon"
- D) Fehler

Richtige Antwort: B) "nohtyP" – [::-1] kehrt
den String um

Blitzantwort (Strings): [:: -1] kehrt den String um

- `s[:: -1]`: Umkehrt das Zeichenarray
- Beispiel: `"abc"[:: -1] => "cba"`

String-Slicing (`s[1:4]`, negatives Slicing)

```
word = "Python"
print(word[1:4])      # "yth"
print(word[-3:])     # "hon"
print(word[::-1])    # "nohtyP"
```

String-Formatierung (f-Strings, .format(), %-Operator)

- f-Strings:

```
name = "Alice"  
age = 30  
print(f"{name} ist {age} Jahre alt.")
```

- .format()-Methode:

```
print("{} ist {} Jahre alt.".format(name, age))
```

Challenge: Zeichen ersetzen (`.replace()`), Wörter zählen (`.split()`)

Aufgabe:

- Ersetze “Welt” durch “Python”
- Zähle die Wörter im Satz

```
text = "Hallo, Welt!"  
text = text.replace("Welt", "Python")  
words = text.split()
```

```
print(text)      # "Hallo, Python!"  
print(len(words)) # 2
```

Vergleich: Tuples, Dictionaries und Strings (Tabelle mit Unterschieden)

Eigenschaft	Tuple	Dictionary	String
Veränderlich?	Nein	Ja	Ja
Schlüssel-Wert?	Nein	Ja	Nein
Geordnet?	Ja	Ja	Ja

Eigenschaft

Tuple

Dictionary

String

Anwendung

Feste
Daten

Zuordnungen

Texte

