Low-cost-Objekte statt Dictionaries?

Detlef Lannert
PyDdf 02. 10. 2924

Sind dicts was Schlechtes?

- nein, aber …
- verschachtelte dict-Zugriffe können unübersichtlich werden:

  ```
  data["result"]["left"][0]["colour"]
  ```

- Lesbarkeit und Schreibaufwand sind unbefriedigend
- Bedeutung von dict-Einträgen ist schlecht zu dokumentieren
- TypedDict: viel Aufwand, wenig Nutzen
- wäre `data.result.left[0].colour` besser schreib- und lesbar?

Alternative 0: dict

```
def get_remote() -> dict[str, Any]:
    return {"host": "192.168.42.3", "port": 8100, "user": "hugo"}
```

- ⊕ immer verfügbar
- ⊕ effizient
- ⊕ besser und stabiler als Tupel
- ⊖ nicht sehr robust
- ⊖ nicht gut dokumentier- und prüfbar

Alternative 1: normale Klasse

```
class HostInfo:
    def __init__(self, *, host: str, port: int, user: str):
        self.host = host
        self.port = port
        self.user = user
hi = HostInfo(host="192.168.42.3", port=8100, user="hugo")
```

- ⊕ gut zu dokumentieren und zu checken
- ⊕ gut erweiterbar
- ⊖ aufwendig zu schreiben
- ⊖ schlechte Darstellung in str und repr

```
<__main__.HostInfo at 0x7fb727e1be90>
```

Alternative 2: named tuple

```
from collections import namedtuple
HostInfo = namedtuple("HostInfo", ["host", "port", "user"])
…
hi = HostInfo(host="192.168.42.3", port=8100, user="hugo")
print(hi.host, hi.port)
```

- ⊕ in der Standardbibliothek
- ⊕ gute Repräsentation:

  ```
  HostInfo(host='192.168.42.3', port=8100, user='hugo')
  ```

- ⊖ ist ein Tupel, also auch als Sequenz verwendbar
- ⊖ keine Methoden möglich

Alternative 3: leere Klasse

```
class HostInfo: pass
hi = HostInfo()
hi.host = "192.168.42.3"
hi.port = 8100
hi.user = "hugo"
```

- ⊕ geringer Schreibaufwand für „Wegwerfklassen“
- ⊖ kann unübersichtlich werden (verteilte Definition)
- ⊖ schlechte Darstellung in str und repr

Alternative 4: dataclass / attrs

```
from attrs import define
@define(kw_only=True)
class HostInfo:
    host: str
    port: int
    user: str
hi = HostInfo(host="192.168.42.3", port=8100, user="hugo")
```

- ⊕ viele nützliche Methoden vordefiniert: `__init__`, str / repr

  ```
  HostInfo(host='192.168.42.3', port=8100, user='hugo')
  ```

- ⊕ type checking
- ⊕ slots voreingestellt, Validatoren / Konverter möglich
- ⊖ für Wegwerfklassen etwas aufwendig

Alternative 5: SimpleNamespace

```
from types import SimpleNamespace as SN
hi = SN(host="192.168.42.3", port=8100, user="hugo")
print(hi.host, hi.port)
```

- ⊕ minimaler Schreibaufwand
- ⊕ nützliche Darstellung

  ```
  namespace(host='192.168.42.3', port=8100, user='hugo')
  ```

- ⊕ gut auf „richtige Klassen“ zu migrieren
- ⊕ in C implementiert
- ⊖ kein type checking
- ⊖ keine Methoden

Fazit?

- one size doesn't fit all
- Objektnotation ist oft besser zu lesen und zu schreiben als Dictionary-Notation
- die verschiedenen klassenbasierten (und ähnlichen) Lösungen sind untereinander ganz gut austauschbar
- De-/Serialisierung, Prüfung und Konvertierung kann bei dataclasses / attrs am einfachsten hinzugefügt werden