



NotImplementedError

```
NotImplementedError: No id attribute - override get_id  
Traceback (most recent call last)
```

```
File "C:\Users\Malin Schütz\vsc-workspace\web entwicklung\Helperbaer\venv\Lib\site-  
packages\flask\app.py", line 1536, in call  
return self.wsgi_app(environ, start_response)  
^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
File "C:\Users\Malin Schütz\vsc-workspace\web entwicklung\Helperbaer\venv\Lib\site-  
packages\flask\app.py", line 1514, in wsgi_app  
response = self.handle_exception(e)  
^^^^^^^^  
File "C:\Users\Malin Schütz\vsc-workspace\web entwicklung\Helperbaer\venv\Lib\site-  
packages\flask\app.py", line 1511, in wsgi_app  
response = self.full_dispatch_request()  
^^^^^^^^  
File "C:\Users\Malin Schütz\vsc-workspace\web entwicklung\Helperbaer\venv\Lib\site-  
packages\flask\app.py", line 919, in full_dispatch_request  
rv = self.handle_user_exception(e)  
^^^^  
File "C:\Users\Malin Schütz\vsc-workspace\web entwicklung\Helperbaer\venv\Lib\site-  
packages\flask\app.py", line 917, in full_dispatch_request  
rv = self.dispatch_request()  
^^^^  
File "C:\Users\Malin Schütz\vsc-workspace\web entwicklung\Helperbaer\venv\Lib\site-  
packages\flask\app.py", line 902, in dispatch_request  
return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args) # type: ignore[no-any-  
return]  
^^^^  
File "C:\Users\Malin Schütz\vsc-workspace\web entwicklung\Helperbaer\app.py", line 78, in  
helfer_anmelden  
login_user(user) #Quelle: Flask-Login, Abschnitt "Login Example"  
^^^^  
File "C:\Users\Malin Schütz\vsc-workspace\web entwicklung\Helperbaer\venv\Lib\site-  
packages\flask_login\utils.py", line 183, in login_user  
user_id = getattr(user, current_app.login_manager.id_attribute)()  
^^^^  
File "C:\Users\Malin Schütz\vsc-workspace\web entwicklung\Helperbaer\venv\Lib\site-  
packages\flask_login\mixins.py", line 27, in get_id
```

```
raise NotImplementedError("No id attribute - override get_id") from None
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

NotImplementedError: No id attribute - override get_id

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

dump() shows all variables in the frame

dump(obj) dumps all that's known about the object. Was bedeutet diese Fehlermeldung

Die Fehlermeldung "NotImplementedError: No id attribute - override get_id" tritt auf, weil Flask-Login beim Aufruf von `login_user(user)` versucht, die ID des User-Objekts zu ermitteln, aber dein User-Modell diese nicht korrekt bereitstellt.^[1] ^[2]

Ursache

Flask-Login erwartet von jedem User-Objekt eine `get_id()`-Methode, die eine eindeutige, serialisierbare ID (meist `str(self.id)`) zurückgibt. Diese stammt standardmäßig aus `UserMixin`, wird aber nicht gefunden, wenn dein Modell kein Primary-Key-Feld `id` hat oder `UserMixin` fehlt.^[3]
^[1]

Lösungsschritte

- **User-Modell erweitern:** Erbe von `flask_login.mixins.UserMixin` und definiere ein `id`-Feld als Primary Key (z. B. mit SQLAlchemy: `id = Column(Integer, primary_key=True)`). Füge optional `def get_id(self): return str(self.id)` hinzu.^[2] ^[3]
- **User-Loader definieren:** Registriere `@login_manager.user_loader def load_user(user_id): return User.query.get(int(user_id))` – das lädt den User anhand der gespeicherten ID aus der Session.^[4] ^[1]
- **Login-Route prüfen:** Stelle sicher, dass `user = User.query.filter_by(...).first()` ein gültiges Objekt mit ID zurückgibt, bevor `login_user(user)` aufgerufen wird.^[5]

Nach diesen Änderungen und Neustart funktioniert `login_user` in deiner `helper_anmelden`-Route (Zeile 78).^[1]

**

import click

```
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from datetime import date, timedelta
```

```

from flask_bcrypt import Bcrypt
from app import app

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///helperbaer.sqlite'

bcrypt = Bcrypt(app)
db = SQLAlchemy()
db.init_app(app)

class User(db.Model, UserMixin): #Quelle UserMixin: Flask-Login, Abschnitt "Your User Class"
    tablename = "user"
    userId = db.Column(db.Integer, primary_key=True, index=True)
    name = db.Column(db.String, nullable=False)
    firstName = db.Column(db.String, nullable=False)
    birthday = db.Column(db.Date, nullable=False)
    email = db.Column(db.String, nullable=False, unique=True)
    password = db.Column(db.String, nullable=False)
    phone = db.Column(db.String)
    role = db.Column(db.String, nullable=False)
    registered_date = db.Column(db.Date, default=date.today())

jobs_created = db.relationship(
    "Job",
    foreign_keys='Job.kundId',
    back_populates="kunde"
)

jobs_taken = db.relationship(
    "Job",
    foreign_keys='Job.helperId',
    back_populates="helper"
)

```

) meine class User sieht bisher so aus. Brauche ich also eine def get_id?

Nein, du brauchst keine explizite def get_id(self)-Methode, wenn du UserMixin importierst und korrekt konfigurierst – der Fehler liegt am fehlenden user_loader und möglicherweise am Primary-Key-Namen. [29] [30]

Problemursachen

UserMixin sucht standardmäßig nach einem Attribut namens id für get_id(), aber dein Primary Key heißt userId. Flask-Login wirft den Fehler, weil login_user die ID nicht findet. Zusätzlich fehlt der @login_manager.user_loader-Callback, der den User aus der Session lädt. [31] [29]

Schnelle Fixes

Füge in app.py (oder wo LoginManager initialisiert wird) hinzu:

```

from flask_login import LoginManager
login_manager = LoginManager()

```

```

login_manager.init_app(app)
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id)) # Lädt per userId (PK)

```

Das passt zu deinem userId-Feld, da SQLAlchemy `query.get()` den Primary Key verwendet. [\[30\]](#) [\[31\]](#)

User-Modell-Optimierung

- Um Konflikte zu vermeiden, benenne den Primary Key zu id um: `id = db.Column(db.Integer, primary_key=True)`. Passe Relationships an (`foreign_keys='Job.kundeId' → 'Job.kunde_id'`). [\[32\]](#) [\[30\]](#)
- Oder überschreibe explizit:

```

def get_id(self):
    return str(self.userId)

```

Das erzwingt Kompatibilität mit `UserMixin`. [\[33\]](#) [\[34\]](#)

Nach Migration (`db.create_all()`) und Neustart löst sich der Fehler in `login_user(user)`. Teste mit `current_user.is_authenticated`. [\[29\]](#)

**

perfekt danke

Gerne! Viel Erfolg mit Helferbär und der Authentifizierung. [\[44\]](#)

**

import click

```

from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from datetime import date, timedelta
from flask_bcrypt import Bcrypt
from app import app

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///helferbaer.sqlite'

bcrypt = Bcrypt(app)
db = SQLAlchemy()
db.init_app(app)

class User(db.Model, UserMixin): #Quelle UserMixin: Flask-Login, Abschnitt "Your User Class"
    tablename = "user"
    userId = db.Column(db.Integer, primary_key=True, index=True)

```

```

name = db.Column(db.String, nullable=False)
firstName = db.Column(db.String, nullable=False)
birthday = db.Column(db.Date, nullable=False)
email = db.Column(db.String, nullable=False, unique=True)
password = db.Column(db.String, nullable=False)
phone = db.Column(db.String)
role = db.Column(db.String, nullable=False)
registered_date = db.Column(db.Date, default=date.today())

jobs_created = db.relationship(
    "Job",
    foreign_keys='Job.kundId',
    back_populates="kunde"
)

jobs_taken = db.relationship(
    "Job",
    foreign_keys='Job.helferId',
    back_populates="helfer"
)

def get_id(self): #Quelle: KI Prompt 2
    return str(self.userId)

#lädt Stunden des aktuellen Monats für Stundenkonto Kunde
@property
def current_month_hours_kunde(self):
    current_month = date.today().replace(day=1)
    next_month = (current_month + timedelta(days=32)).replace(day=1)
    jobs = [job for job in self.jobs_created if current_month <= job.date < next_month]

    offene = sum(job.hours for job in jobs if job.statusId == 1)
    gebuchte = sum(job.hours for job in jobs if job.statusId == 2)
    erledigte = sum(job.realHours or job.hours for job in jobs if job.statusId == 3)

    return {
        'offene': offene,
        'gebuchte': gebuchte,
        'erledigte': erledigte,
        'gesamt': offene + gebuchte + erledigte
    }

#lädt Stunden des aktuellen Monats für Stundenkonto Helfer
@property
def current_month_hours_helfer(self):
    current_month = date.today().replace(day=1)
    next_month = (current_month + timedelta(days=32)).replace(day=1)

```

```

jobs = [job for job in self.jobs_taken if current_month <= job.date < next_month]

gebuchte = sum(job.hours for job in jobs if job.statusId == 2)
erledigte = sum(job.realHours or job.hours for job in jobs if job.statusId == 3)

return {
    'gebuchte': gebuchte,
    'erledigte': erledigte,
    'gesamt': gebuchte + erledigte
}

#angefragte und erledeigte Jobs für Kunden Dashboard
def get_jobs_by_status_kunde(self):
    jobs = Job.query.filter(
        Job.kundId == self.userId
    ).order_by(Job.date.desc()).all()

    # Trennung: angefragt (offen/gebucht) vs erledigt
    angefragte_jobs = [j for j in jobs if j.statusId in [1,2]]
    erledigte_jobs = [j for j in jobs if j.statusId == 3]

    return {
        'angefragte_jobs': angefragte_jobs,
        'erledigte_jobs': erledigte_jobs
    }

#angefragte und erledeigte Jobs für Helper Dashboard
def get_jobs_by_status_helper(self):
    jobs = Job.query.filter(
        Job.helperId == self.userId
    ).order_by(Job.date.desc()).all()

    # Trennung: gebuchte vs erledigt
    gebuchte_jobs = [j for j in jobs if j.statusId == 2]
    erledigte_jobs = [j for j in jobs if j.statusId == 3]

    return {
        'gebuchte_jobs': gebuchte_jobs,
        'erledigte_jobs': erledigte_jobs
    }

#Stundenkonto für Helper Dashboard
@property
def gesamtArbeitsStunden(self):
    if self.role != 'helper':
        return 0

```

```

return sum(
    job.realHours or 0
    for job in self.jobs_taken
    if job.statusId == 3
) warum haben manche funktionen @property decorator und ist das zwingend notwendig?

```

Der `@property`-Decorator macht Methoden zu **read-only Attributen**, die wie normale Attribute aufgerufen werden (ohne Klammern). Er ist nicht zwingend notwendig, aber bei deinen Stundenkonto-Funktionen sehr sinnvoll.

Wann `@property` verwenden

- `current_month_hours_kunde` und `current_month_hours_helper`: `@property` → Zugriff als `user.current_month_hours_kunde` (Dict zurück).
- **Ohne** `@property`: `user.current_month_hours_kunde()` (Funktionsaufruf).
- **Templates/Jinja**: `{{ current_user.current_month_hours_kunde.offene }}` funktioniert nur mit `@property`.^[45]

Deine Methoden-Analyse

```

@property          # ← Wird wie Attribut behandelt
def current_month_hours_kunde(self):
    return {...}      # → user.current_month_hours_kunde['offene']

def get_jobs_by_status_kunde(self):  # ← Normale Methode (kein @property)
    return {...}      # → user.get_jobs_by_status_kunde()['angefragte_jobs']

```

Funktion	<code>@property</code>	Aufruf	Sinnvoll für
<code>current_month_hours_*</code>	✓ Ja	<code>user.zeitkonto</code>	Templates, automatische Berechnung
<code>get_jobs_by_status_*</code>	✗ Nein	<code>user.get_jobs()</code>	Expliziter Aufruf, komplexe Logik
<code>gesamtArbeitsStunden</code>	✓ Ja	<code>user.gesamtStunden</code>	Dashboard-Anzeige

Empfehlung

- **Behalte** `@property` bei Stundenkonten – perfekt für Templates.
- **Normale Methoden** für Job-Listen (expliziter, parameterisierbar).
- **Performance**: Cache `@property` bei schweren Berechnungen mit `functools.lru_cache`.

```

@property
@lru_cache(maxsize=1)
def current_month_hours_kunde(self):
    # Dein Code...

```

Das ist sauberes, idiomatisches Python für dein Helferbär-Dashboard!^[46]

gibt es in einer flask Dokumentation was zu den properties

Nein, die offizielle Flask-Dokumentation erwähnt @property nicht explizit, da es ein **reines Python-Feature** ist, kein Flask-spezifisches Konzept.

Python-Dokumentation (offiziell)

@property wird in der Python-Dokumentation beschrieben:

```
https://docs.python.org/3/library/functions.html#property
https://docs.python.org/3/tutorial/classes.html#properties
```

```
class C:
    def __init__(self):
        self._x = None

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, value):
        self._x = value
```

Flask-Kontext

Flask-Dokumentation behandelt **andere Decorators** wie @app.route, @login_required:

```
https://flask.palletsprojects.com/en/stable/patterns/viewdecorators/
```

```
@app.route('/')
@login_required
def index():
    return "Hallo"
```

Dein User-Modell

```
@property # ← Python built-in, nicht Flask
def current_month_hours_kunde(self):
    return {...} # → user.current_month_hours_kunde['offene']
```

Fazit: @property kommt aus Python-Docs, nicht Flask-Docs. Perfekt für Jinja-Templates in Flask ({{ current_user.gesamtStunden }}).

gibt es ein tutorial für properties?

Ja, hier sind die besten deutschsprachigen Tutorials für @property:

Top-Empfehlungen

1. Python-Basics Tutorial (schriftlich)

<https://python-basics-tutorial.readthedocs.io/de/24.3.0/oop/property.html>

- Perfekte Erklärung mit Getter/Setter-Beispielen
- Beispiele: `s1.length = 2` → `@length.setter`
- **Ideal für Helferbär:** Zeigt Klassen-Attribute wie deine Stundenkonten

2. YouTube-Videos (praktisch)

- "Property Decorator: GETTER, SETTER und DELETER" (11 Min)
<https://www.youtube.com/watch?v=PgekwucmID8>
- "Getters & Settings (@Property Decorator)" (OOP-Serie)
<https://www.youtube.com/watch?v=CWd2USTgc8Q>

Praktisches Beispiel (für dein User-Modell)

```
class User(db.Model, UserMixin):
    # ... deine Felder ...

    @property
    def full_name(self): # ← Getter
        return f"{self.firstName} {self.name}"

    @full_name.setter # ← Setter
    def full_name(self, value):
        names = value.split(" ", 1)
        self.firstName, self.name = names[0], names[1]

    # Nutzung:
    user.full_name # → "Max Mustermann" (ruft Getter)
    user.full_name = "Anna Beispiel" # ruft Setter
```

Start mit dem Python-Basics-Link – passt perfekt zu deinem Flask/SQLAlchemy-Code-Stil!

könnte ich auch ohne property ein dictionary zurückgeben?

Ja, absolut! Du kannst problemlos ein Dictionary ohne @property zurückgeben.

Ohne @property (normale Methode)

```
def current_month_hours_kunde(self):  # ← Kein @property
    # Dein Code identisch...
    return {'offene': 5, 'gebuchte': 3, 'erledigte': 8, 'gesamt': 16}

# Aufruf:
user.current_month_hours_kunde()  # ← Mit Klammern!
result = user.current_month_hours_kunde()
print(result['offene'])  # 5
```

Mit @property (Attribut)

```
@property
def current_month_hours_kunde(self):
    # Gleicher Code...
    return {'offene': 5, 'gebuchte': 3, 'erledigte': 8, 'gesamt': 16}

# Aufruf:
user.current_month_hours_kunde  # ← Ohne Klammern!
print(user.current_month_hours_kunde['offene'])  # 5
```

Wann welches verwenden?

Kontext	@property	Normale Methode
Templates	✓ {{ current_user.stunden['offene'] }}	✗ {{ current_user.stunden()['offene'] }}
Python-Code	✓ user.stunden['gesamt']	✓ user.stunden()['gesamt']
Parameter	✗	✓ user.stunden(2024, 1)
Performance	⚠ Neu berechnet bei jedem Zugriff	✓ Kann gecacht werden

Deine beste Wahl für Helferbär

```
# Für Templates (Dashboard) → @property behalten
@property
def current_month_hours_kunde(self):
    return {...}

# Für komplexe Abfragen → normale Methode
```

```
def get_hours_for_month(self, year, month):
    return {...} # Mit Parametern!
```

Fazit: Dictionary-Rückgabe klappt immer. @property nur für **häufige Template-Zugriffe** sinnvoll.

*

1. <https://stackoverflow.com/questions/73977948/flask-login-using-usermixin-no-id-attribute-error-despite-being-implemented>
2. <https://flask-login.readthedocs.io>
3. <https://www.freecodecamp.org/news/how-to-authenticate-users-in-flask/>
4. <https://www.digitalocean.com/community/tutorials/how-to-add-authentication-to-your-app-with-flask-login>
5. https://flask-login.readthedocs.io/en/0.4.0/_modules/flask_login/utils.html
6. https://flask-login.readthedocs.io/en/0.6.2/_modules/flask_login/mixins/
7. <https://faun.pub/python-flask-tutorial-part-5-d3f0be695dac>
8. <https://github.com/techwithtim/Flask-Web-App-Tutorial/issues/84>
9. https://www.reddit.com/r/flask/comments/as1jiz/af_user_object_has_no_attribute_is_active/
10. <https://gist.github.com/shrayasr/10004888>
11. <https://www.freekb.net/Article?id=4677>
12. <https://github.com/pallets-eco/flask-security-3.0/issues/472>
13. <https://flask123.sinaapp.com/article/13/>
14. <https://github.com/mattupstate/flask-security/issues/472>
15. <https://www.mitchellbusby.com/2016/08/21/tutorial-roll-your-own-user-authorisation-management-with-flask-login/>
16. <https://stackoverflow.com/questions/10695093/how-to-implement-user-loader-callback-in-flask-login>
17. https://flask-user.readthedocs.io/en/latest/data_models.html
18. https://www.reddit.com/r/flask/comments/1bifyui/dont_understand_login_manager_and_user_loader/
19. <https://www.freecodecamp.org/news/how-to-setup-user-authentication-in-flask/>
20. https://www.reddit.com/r/flask/comments/1adr569/use_flasklogin_without_sqlalchemy/
21. <https://stackoverflow.com/questions/12075535/flask-login-cant-understand-how-it-works>
22. https://github.com/maxcountryman/flask-login/blob/main/src/flask_login/utils.py
23. <https://hackersandslackers.com/flask-login-user-authentication/>
24. https://www.reddit.com/r/flask/comments/l37x94/can_someone_help_me_understand_user_loader/
25. <https://stackoverflow.com/questions/29199624/flask-login-errors-with-specific-attributes>
26. <https://stackoverflow.org.cn/questions/57893830>
27. <https://www.appspotworld.com/coding/flask/72/flask-login-get-id-override-prevents-other-methods-from-working>
28. <https://stackoverflow.com/questions/67964061/flask-login-get-id-override-prevents-other-methods-from-working>
29. <https://flask-login.readthedocs.io>

30. <https://www.digitalocean.com/community/tutorials/how-to-add-authentication-to-your-app-with-flask-login>
31. <https://stackoverflow.com/questions/73977948/flask-login-using-usermixin-no-id-attribute-error-despite-being-implemented>
32. <https://www.freecodecamp.org/news/how-to-setup-user-authentication-in-flask/>
33. <https://github.com/mattupstate/flask-security/issues/472>
34. <https://stackoverflow.com/questions/34951134/flask-login-only-works-if-get-id-returns-self-email>
35. <https://www.mitchellbusby.com/2016/08/21/tutorial-roll-your-own-user-authorisation-management-with-flask-login/>
36. <https://www.freekb.net/Article?id=4545>
37. <https://stackoverflow.com/questions/50138426/how-to-get-name-of-logged-user-flask-sqlalchemy>
38. <https://groups.google.com/g/google-web-toolkit/c/JVaYlul7v6Q>
39. <https://hackersandslackers.com/flask-login-user-authentication/>
40. https://flask-user.readthedocs.io/en/latest/data_models.html
41. <https://overiq.com/flask-101/authentication-in-flask/index.html>
42. https://flask-login.readthedocs.io/_/downloads/en/0.4.0/pdf/
43. <https://www.youtube.com/watch?v=2dEM-s3mRLE>
44. <https://flask-login.readthedocs.io>
45. <https://flask-login.readthedocs.io>
46. <https://www.digitalocean.com/community/tutorials/how-to-add-authentication-to-your-app-with-flask-login>
47. <https://stackoverflow.com/questions/10695093/how-to-implement-user-loader-callback-in-flask-login>
48. https://www.reddit.com/r/flask/comments/tu060b/implementing_different_classes_for_flask_login/
49. <https://flask-login.readthedocs.io/en/latest/>
50. https://www.reddit.com/r/flask/comments/1bifyui/dont_understand_login_manager_and_user_loader/
51. https://www.reddit.com/r/flask/comments/l37x94/can_someone_help_me_understand_user_loader/
52. https://www.reddit.com/r/flask/comments/sg2y2n/2_type_user_login_using_flasklogin/
53. https://www.reddit.com/r/learnpython/comments/h9m9ey/flasklogin_missing_user_loader/
54. <https://gist.github.com/leongjinqwen/a205cbe8185d8c83f9d300cc6c8634f1>
55. <https://flask.palletsprojects.com/en/stable/patterns/viewdecorators/>
56. <https://flask-restplus.readthedocs.io/en/0.7.1/documenting.html>
57. <https://flask.palletsprojects.com/en/stable/api/>
58. <https://flask-restplus.readthedocs.io/en/stable/swagger.html>
59. <https://book.pythontips.com/en/latest/decorators.html>
60. <https://flaskforstartups.com/tutorials/function-decorators-and-routes/>
61. <https://stackoverflow.com/questions/17330160/how-does-the-property-decorator-work-in-python>
62. <https://flask-plugin.pages.dev/api>
63. <https://dev.to/the1kimk/function-decorators-in-python-understanding-property-getter-and-setter-methods-3a8e>
64. <https://flask.palletsprojects.com/en/stable/blueprints/>

65. <https://www.youtube.com/watch?v=Akaoc7BMrfQ>
66. <https://python-basics-tutorial.readthedocs.io/de/24.3.0/oop/property.html>
67. <https://www.youtube.com/watch?v=PgekwucmID8>
68. <https://www.ionos.de/digitalguide/websites/web-entwicklung/python-property/>
69. <https://de.w3docs.com/snippets/python/wie-funktioniert-der-property-decorator-in-python.html>
70. <https://www.youtube.com/watch?v=64VsNQjlx2c>
71. https://www.python-kurs.eu/python3_dekoratoren.php
72. <https://www.ionos.de/digitalguide/websites/web-entwicklung/python-decorators/>
73. <https://www.youtube.com/watch?v=7wKBS831vkM>
74. <https://www.python-forum.de/viewtopic.php?t=11848>
75. https://www.reddit.com/r/learnpython/comments/16msif3/returning_keys_with_matching_values_when_they_keys/
76. <https://marketmix.com/de/python-dictionary-cheat-sheet/>
77. https://www.python-kurs.eu/python3_dictionaries.php
78. https://www.reddit.com/r/Python/comments/d62du1/help_understanding_the_strangeness_of_dict/
79. <https://stackoverflow.com/questions/3137685/using-property-decorator-on-dicts>
80. <https://statistikguru.de/python/python-schlüssel-dictionarys-entfernen.html>
81. <https://wiki.somebot.de/python/dicts/methoden/>
82. <http://py-tutorial-de.readthedocs.io/de/python-3.3/datastructures.html>