Python Osterworkshop 2023

Klassen (und ein bisschen SOLID)

# Warum Klassen?

* Lesbarkeit
* Leicht zu erweitern
* Leicht anzupassen
* Single Responsibility
* Wiederverwendbarkeit

# Thema: Büchereisystem

## Vorgehen erläutern

1. Mögliche Interaktionen und Objekte überlegen
2. Klassen (Objekte) definieren
3. Eigenschaften definieren
4. Tätigkeiten definieren

## Klassen definieren

Klassen = Objekte, Methoden = Verben/Adjektive

Leere Klassen: Book, Library, Member

## Klassen mit Methoden füllen

Alle ohne Variablen, ohne Docstring aber mit pass. Self ansprechen, init erklären.

\_\_init\_\_ method ist der Constructor

\_\_repr\_\_ erklären, kurzer Ausflug in built-ins

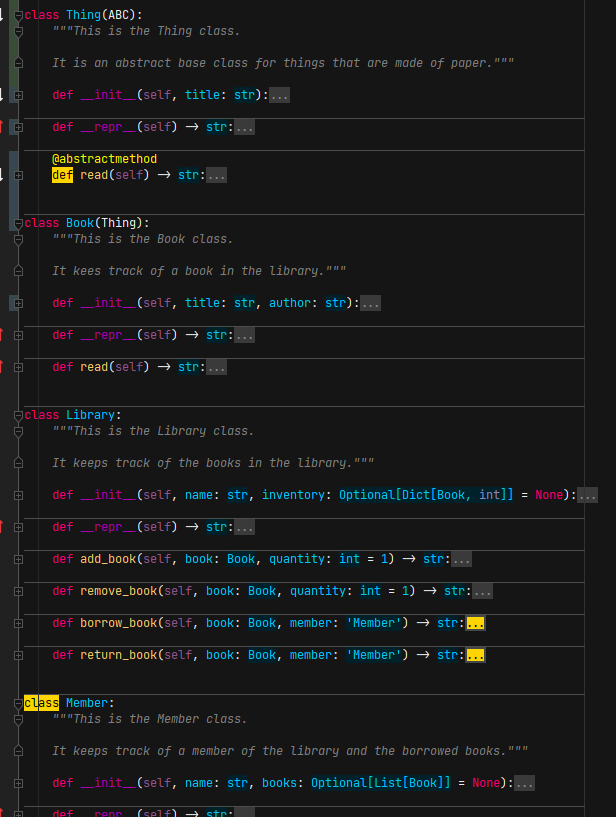
Book: init, repr, read (alle mit pass)

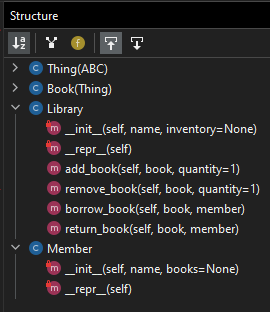
Library: init, repr, add\_book, remove\_book, borrow\_book, return\_book

Member: init, repr

(Code auf nächster Seite)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Single Responsibility | Open/Closed | Liskov Substitution | Interface Segregation | Dependency Inversion |
|  |  |  |  |  |





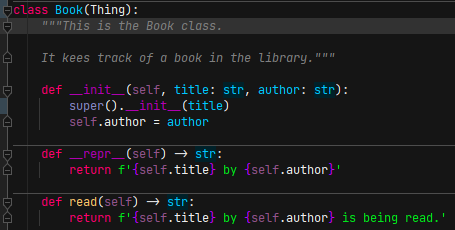
Struktur, mit und ohne inherited

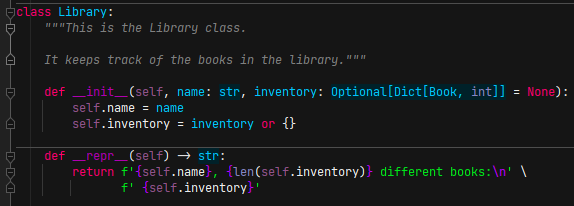
## Init und repr Methoden füllen

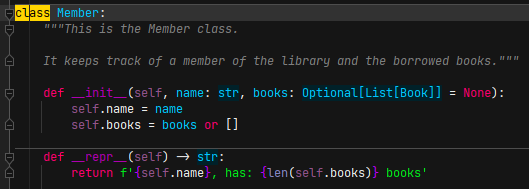
Füllen und Typehints dazu + importieren



**In Book \_\_init\_\_: self.title = title**

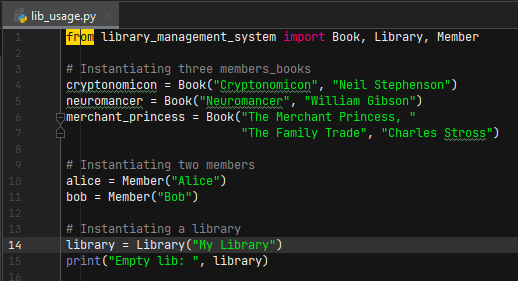






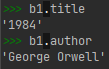
Ausführbare Datei nebendran parallel aufmachen und demonstrieren.

Merchant\_princess.read() usw.

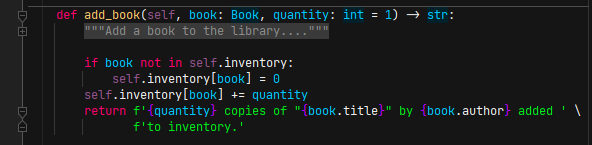


Member und Book sind fertig!

Book.title und book.author demonstrieren.



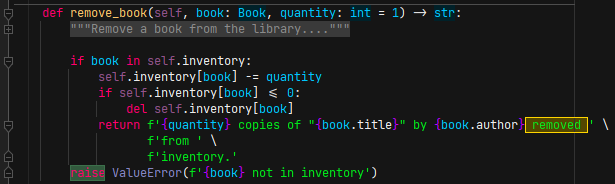
## Library Class – add\_book()



Type hints Book und quantity und return erklären.

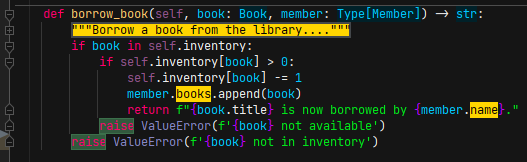
Demo ☺

## Library Class – remove\_book()



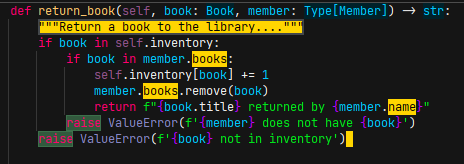
Errors: ValueErrors usw. Fail-Fast (später wird teurer) + Safe-to-fail (erwähnen: Test driven development als Möglichkeit)

## Library Class – borrow\_book()

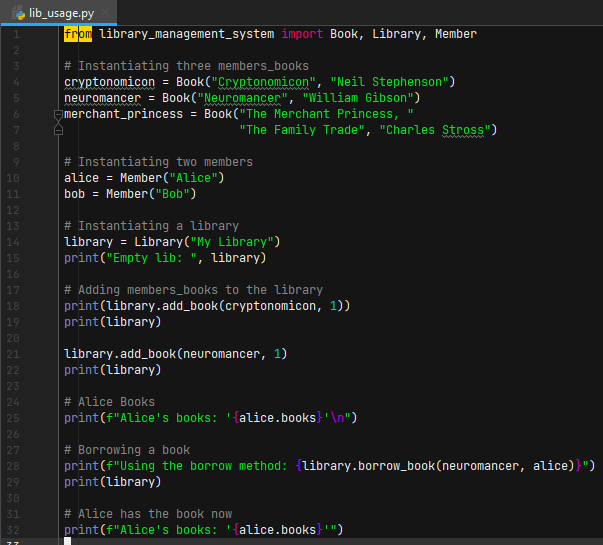


Wenn-dann Fehler. Ist halt eine schöne Verschachtelung ☺

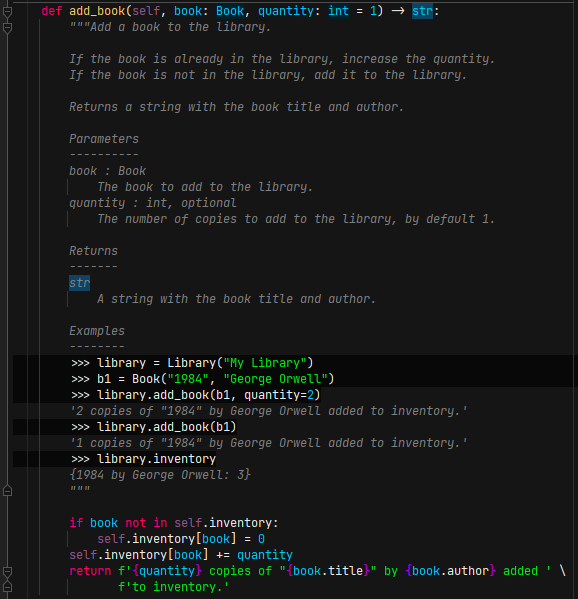
## Library Class – return\_book()

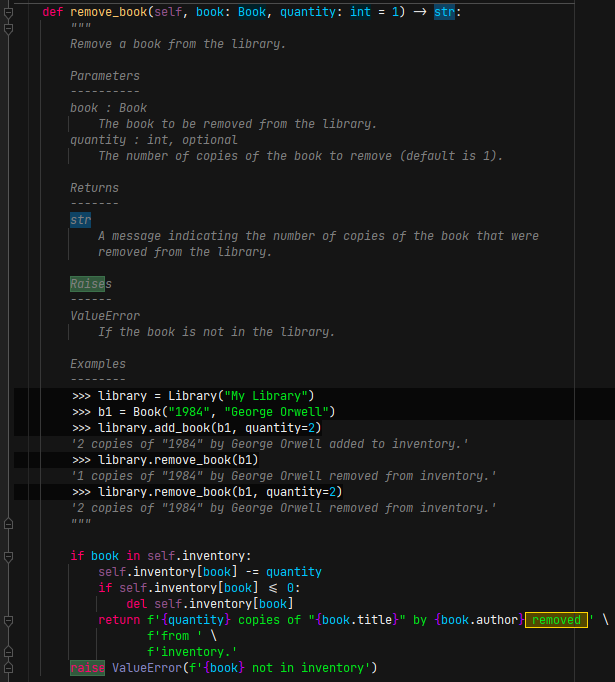


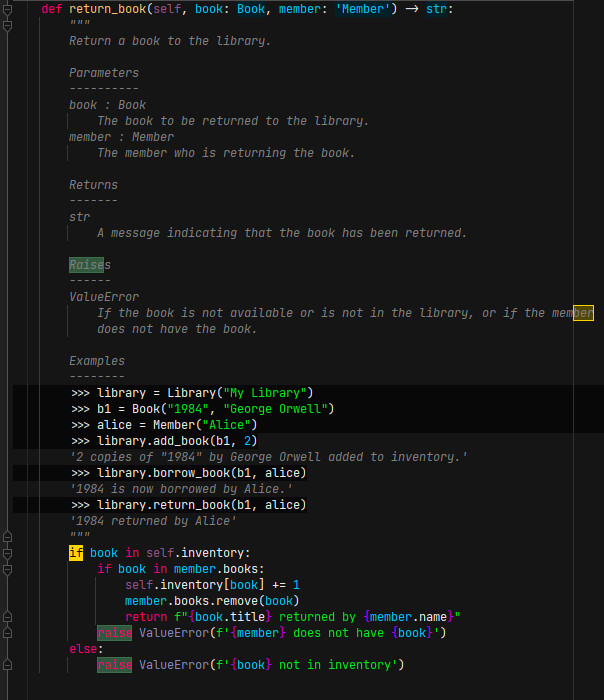
## Demo was bisher funktioniert

Docstring Demo

## Docstrings + Examplels + Example-Test zeigen mit Demo







# Vorteil Dokumentation im Terminal zeigen

Demo

>>> from library\_mgt import Library

>>> alexandria = Library(name=”alex”)

>>> alexandria.\_\_doc\_\_

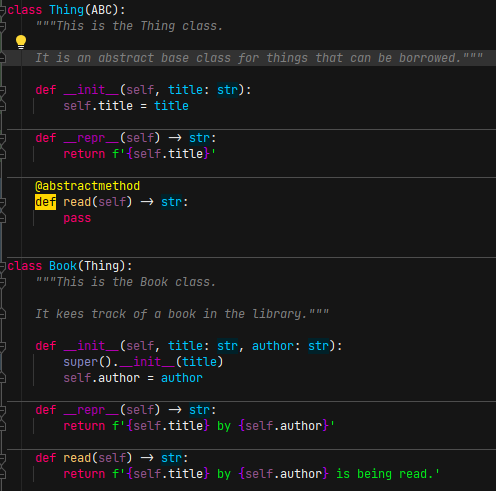
>>> alexandria.add\_book.\_\_annotations\_\_

>>> help(alexandria.return\_book)

# Markdown kurz erklären + Doku auf Website

<https://www.jetbrains.com/help/pycharm/markdown.html>

## ABCs – Erklären von Abstract Base Classes und @abstractmethod + super().\_\_init\_\_



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Single Responsibility | Open/Closed | Liskov Substitution | Interface Segregation | Dependency Inversion |
|  |  |  |  |  |

# Ein Modul bauen

Modul == Pythondatei mit Snake-Case Dateinamen

my\_module.py -> ist ein Modul

Im Code kann man dann

**>>> from my\_module import MyClass**

Ein Package bauen

Wie oben, bloss dass man mehrere Dateien in einen Ordner wirft (eine Klasse pro Datei). In den Ordner kommen noch eine **requirements.txt** eine **readme.md** und eine (meistens leere) **\_\_init\_\_.py** Datei.

**my\_package/**

**\_\_init\_\_.py**

**readme.md**

**requirements.txt**

**my\_module.py**

**my\_other\_module.py**

Im Code von ausserhalb des Verzeichnisses kommt man an die Klassen mit

**>>> from my\_package import my\_module**

**>>> from my\_package.my\_module import MyClass as mc**