JAVACREAM

*Training*
*Consulting*
*Projectmanagement*

# Python

# Kurze Vorstellung

JAVACREAM
Training
Consulting
Projectmanagement

- Name
- Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Aktuelle Problemstellung
- Konkrete individuelle Zielsetzung

# Übersicht

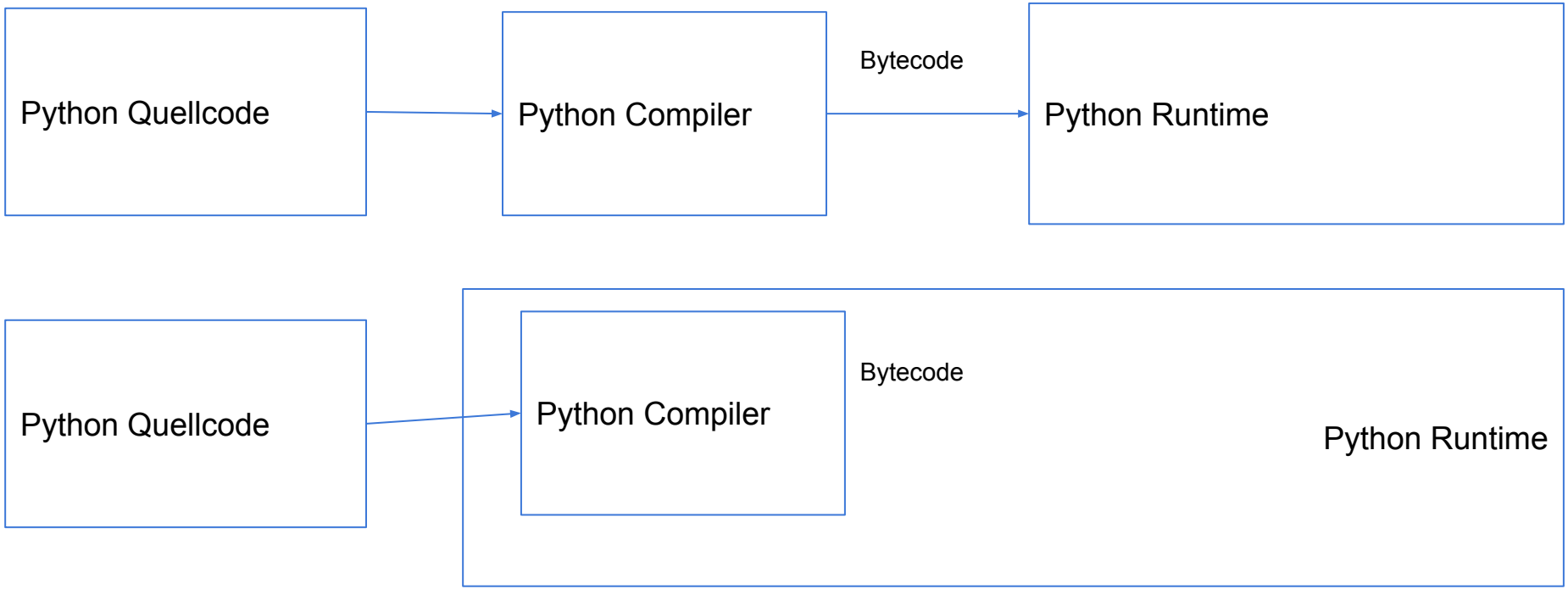
# Was ist Python?



- Python Runtime
  - Häufig genannt “der Python Interpreter”
  - Implementierungen für verschiedene Plattformen / Betriebssysteme sind vorhanden
    - Damit ist Python “Plattform-unabhängig”
    - Es gibt im Gegensatz zu z.B. Java keine Spezifikation
      - Dieser Effekt spielt heute keine große Rolle mehr
- Programmiersprache Python
  - Eigentlich eine “ganz normale” Programmiersprache
  - Einordnung
    - Sequenzielle Programm-Ausführung
    - Laufzeit-Typisiert, optional: eine statische Typisierung ist möglich
    - Objekt-orientierte

# Details zum Python Interpreter





# Infrastruktur

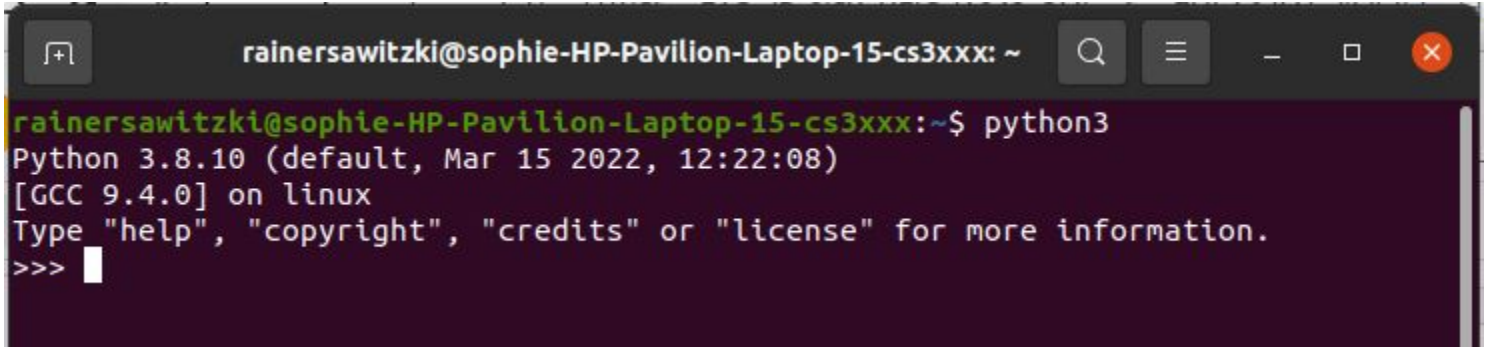


- Installation einer Python Runtime
  - Native Installation auf einem Betriebssystem


  - Verwendung eines Docker-Containers

# Check




rainersawitzki@sophie-HP-Pavilion-Laptop-15-cs3xxx: ~
rainersawitzki@sophie-HP-Pavilion-Laptop-15-cs3xxx:~$ python3
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>

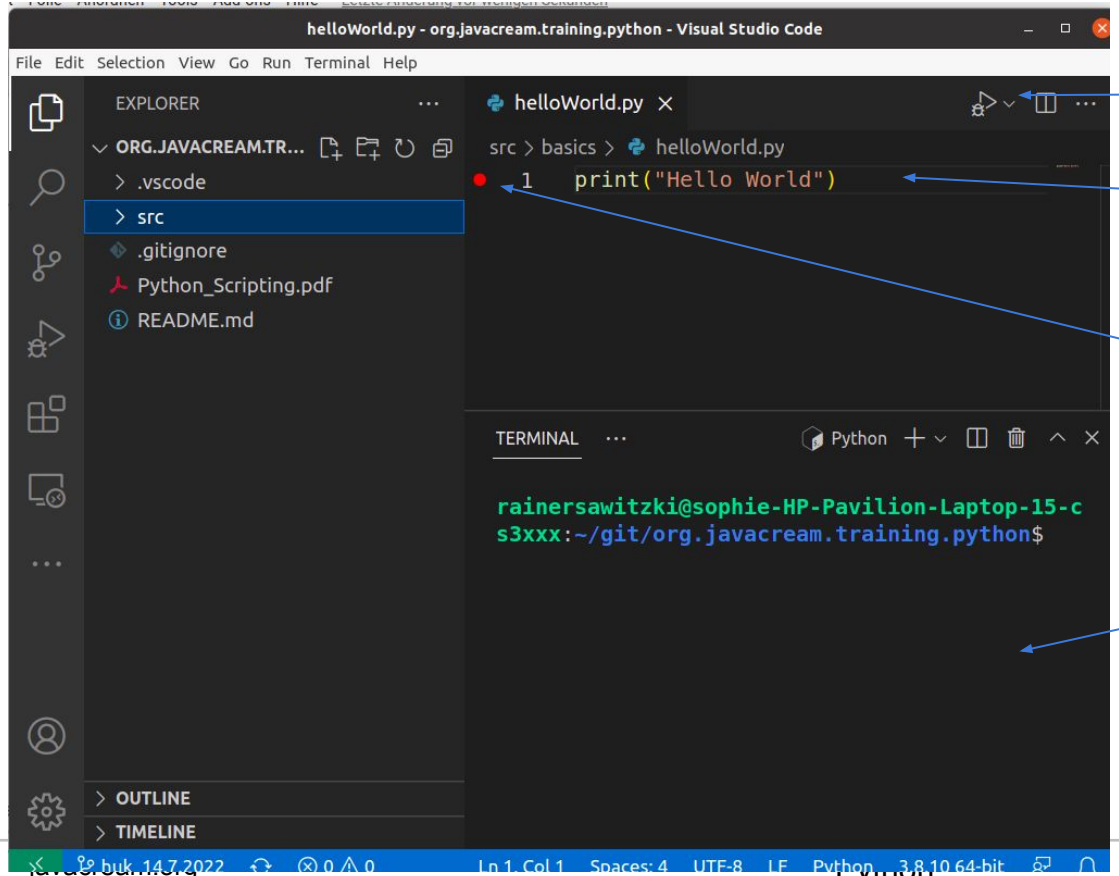
# Python Development



- Entwicklungsumgebung
- **Visual Studio Code**
  - Alternativ: Eclipse-basierte Lösung, Visual Studio, IntelliJ
-

# VSC






Starten / Debuggen eines Python Programms

Editor mit Syntax-Highlighting und Code Assist

Haltepunkt gesetzt “im Quellcode”

Konsole / Terminal

# Python First Contact

# Datentypen



- Literale
  - Zeichenketten
    - "...", '...'
  - Zahlen
    - 9, 4.2, 4.2e-9
  - Logische Werte, Booleans
    - True, False
  - + ein paar mehr…
-

# Python als (fast) normale Programmiersprache



- Operatoren
  - Mathematische Operatoren
    - +, -, *, /, %
  - Zeichenketten-Konkatination
    - +
  - Vergleichs-Operatoren mit logischem Rückgabewert
    - >, <, >=, <=, ==, !=
  - Logische Operatoren
    - and, or, not, is, is not
- Kontrollstrukturen
  - If - Else
    - Arbeitet mit Einrückungen
    - if, else, elif
  - ab Python 3.10: match, das entspricht einem switch-case

# Datentypen continued



- Tupel
  - Eine geordnete, unveränderbare Liste mit erlaubten Duplikaten
  - Literal
    - tuple = (element1, element2, element3)
    - value = tuple[1] # -> element2
    - #tuple[3] = element4 #Fehler, Tupels sind unveränderbar
- List
  - Eine geordnete, veränderbare Liste mit erlaubten Duplikaten
  - Literal
    - list = [element1, element2, element3]
    - value = list[1]
    - list[3] = element4
    - list[1] = element5

# Datentypen continued



- Set
  - Eine ungeordnete, veränderbare Liste ohne Duplikate
  - Literal
    - set = {element1, element2, element3}
    - Zugriff auf Elemente nur über Iteration
- Dictionary
  - Eine ungeordnete, veränderbare Liste mit key=value-Paaren ohne Key-Duplikate
    - dictionary = {“spring”: “ok”, “summer”: “great”, “autumn”: “not so great”, “winter”: “depressive”}
    - Zugriff auf Elemente erfolgt mit dem key
      - value = dictionary[“summer”] #-> great

ab Python 3.x bleibt die Ordnung der keys erhalten

# Speichermodell in Python

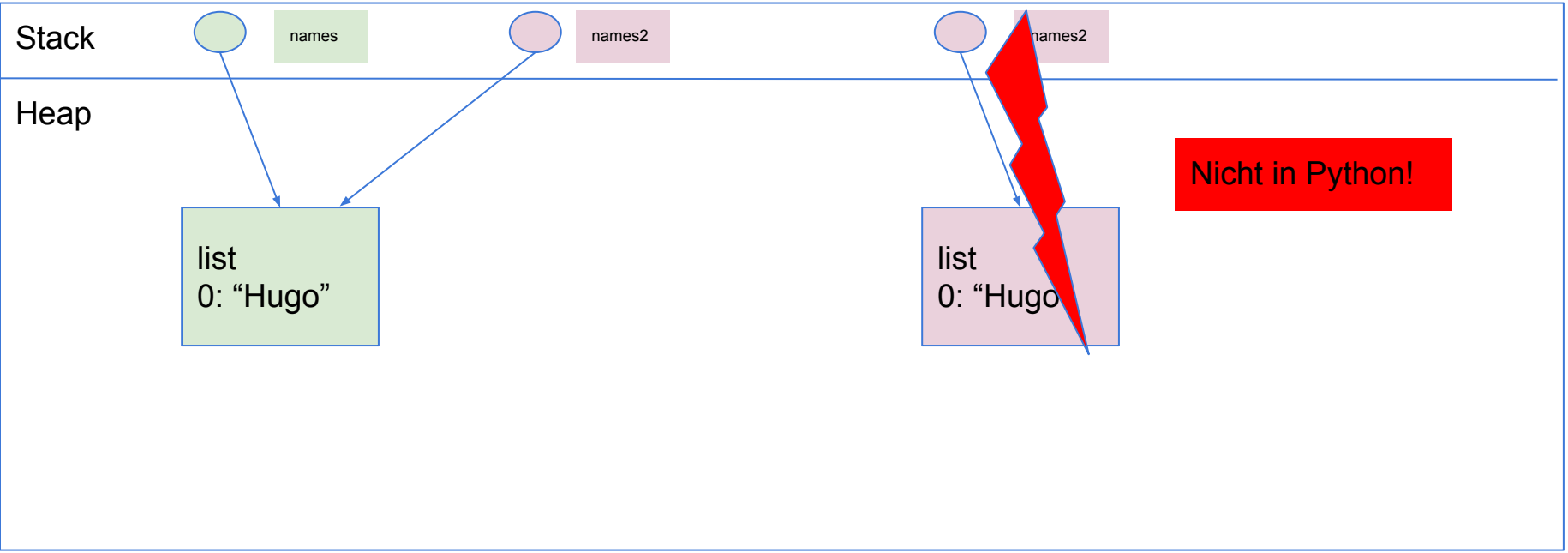


```
names = [“Hugo”]
```

Eine Variable in Python ist immer einer Referenz auf ein Objekt im Heap

```
names2 = names
```

Eine Zuweisung in Python ist immer einer Kopie des Wertes der Referenz

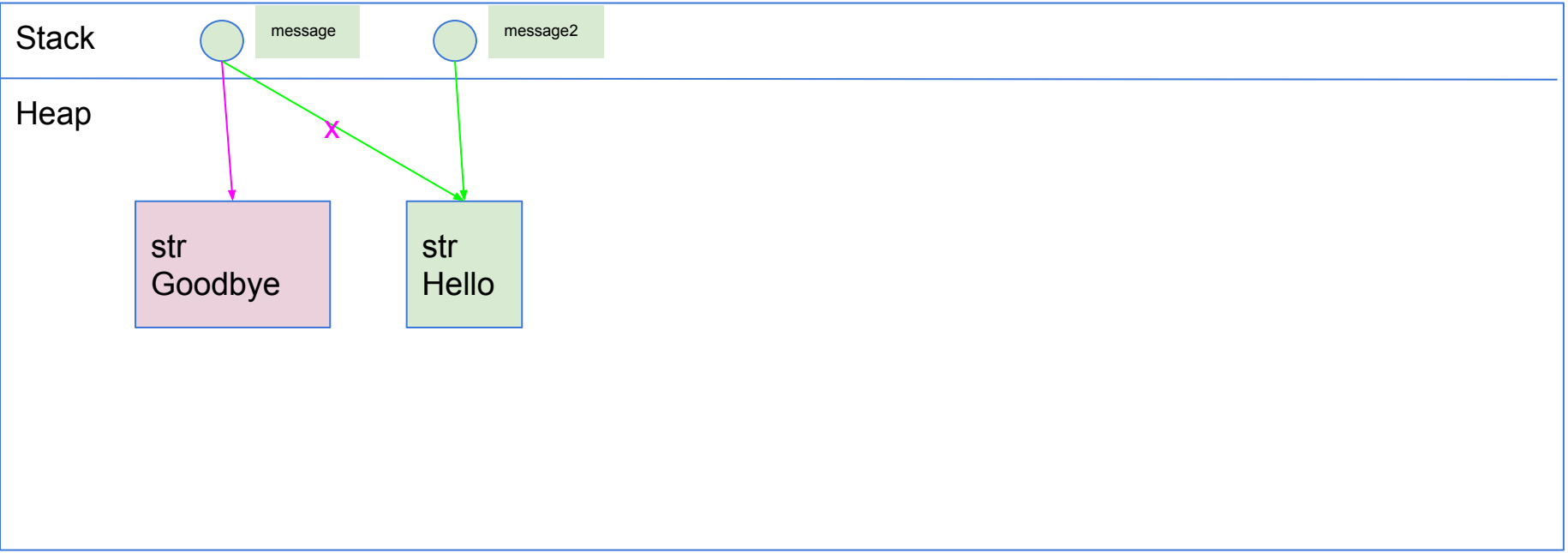


# Speichermodell



```
message = “Hello”
message2 = message
```

```
message = “Goodbye”
```



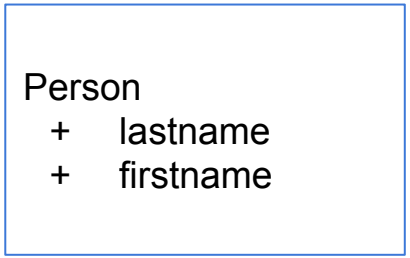
# Namenskonventionen in Python



- Variablen und Funktionsnamen nur Kleinbuchstaben
- Zur besseren Lesbarkeit: snake-Konventionen
  - nicht personname, sondern person_name
    - Andere Sprachen: camel-Case, personName
- Benutzerdefinierte Datentypen (-> etwas später) beginnen mit einem Großbuchstaben und sind Came Case

# Modellierung einer Anwendung



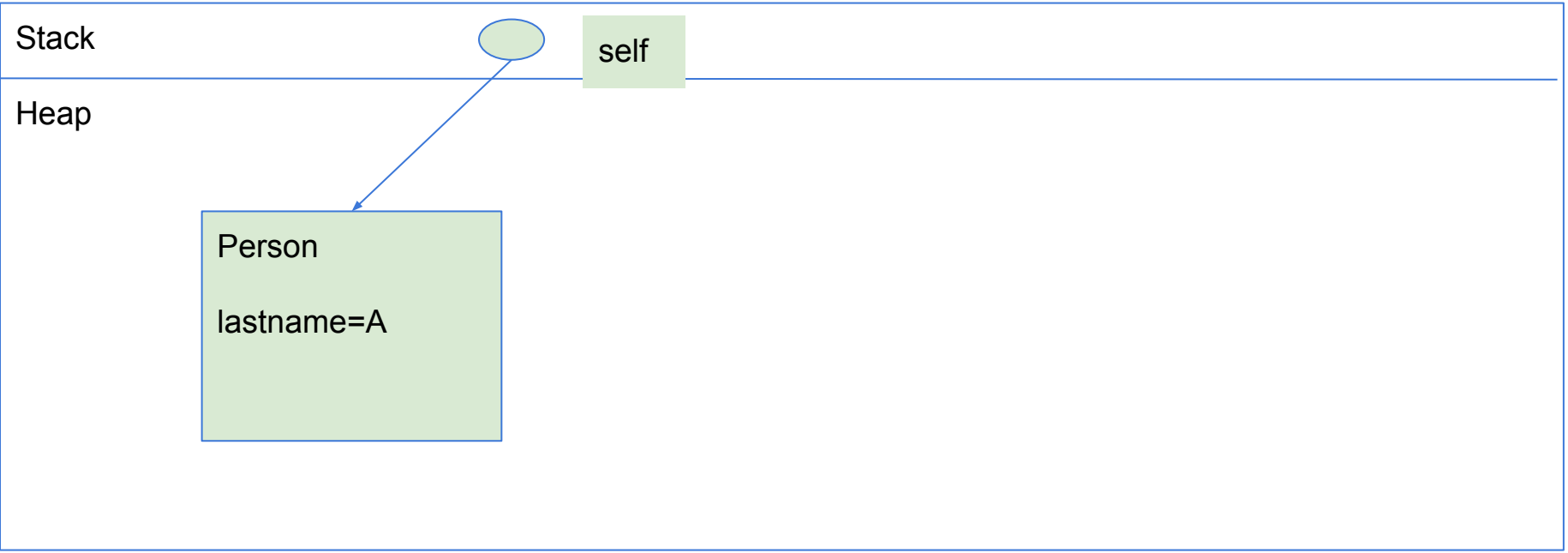


# Speichermodell und self



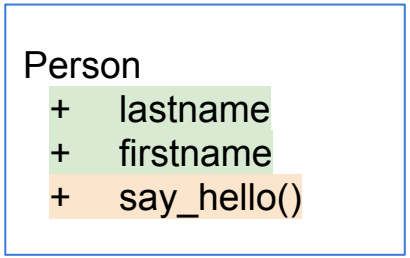
Person(“A”, “B”)

```
__init__(lastname, firstname):
        self.lastname = lastname
```



# Modellierung einer Anwendung



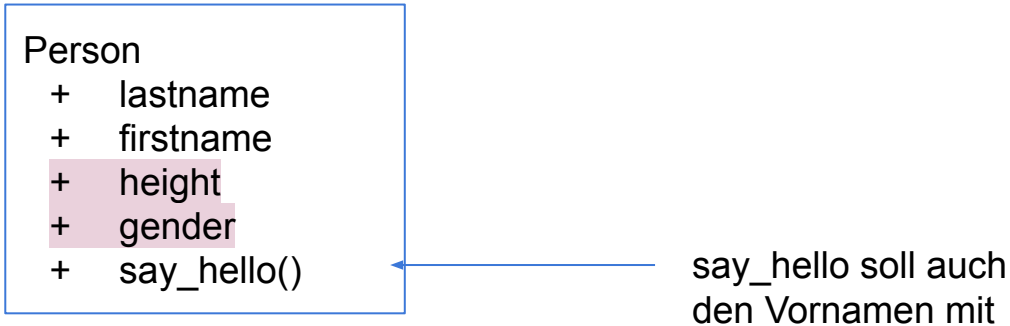



Attribute
Fields

Methods

# Modellierung einer Anwendung

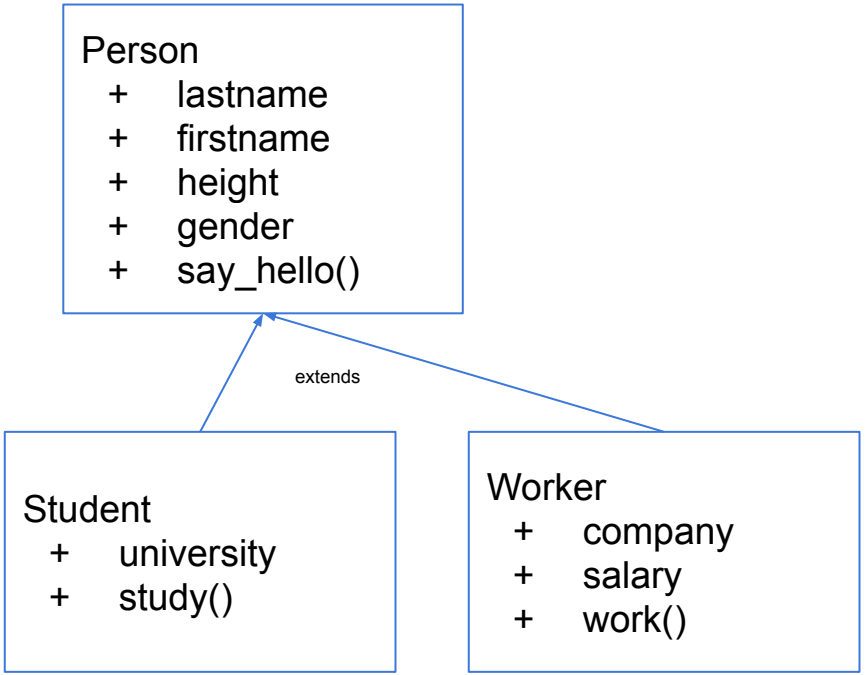





ToDo: Umsetzung dieses Modells

# Vererbung






Python-Scratch

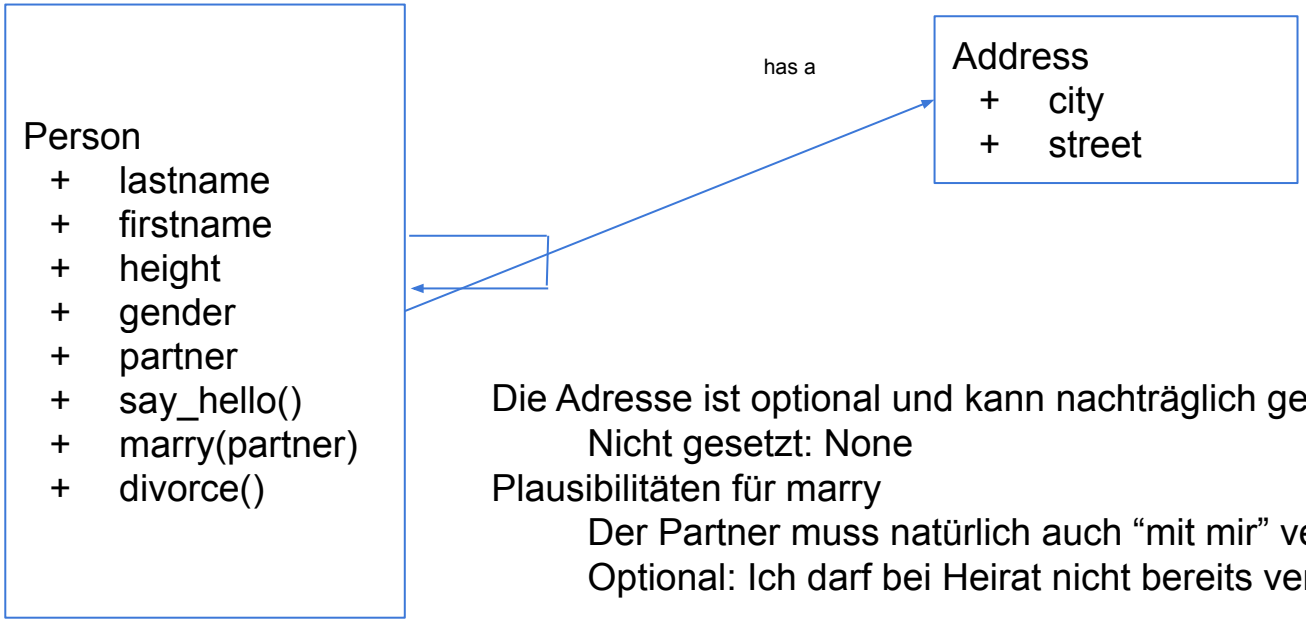
```
class Person:
        …

class Student(Person):
        …

class Worker(Person):
        …
```

# ToDo Extended Modell






Die Adresse ist optional und kann nachträglich gesetzt werden
- Nicht gesetzt: None

Plausibilitäten für marry
- Der Partner muss natürlich auch “mit mir” verheiratet werden
- Optional: Ich darf bei Heirat nicht bereits verheiratet sein
- …

- Wie konsequent ist die Objekt-Orientierung umgesetzt?
  - Arbeiten mit Referenzen ist umgesetzt
    - Verfolgen der Referenzen erfolgt über den '.'-Operator
  - Python-Runtime implementiert einen Garbage Collector
  - (Polymorphes Verhalten ist umgesetzt)
  - "Alles" ist ein Objekt
    - Built-In-Functions von Python haben keinen Bezug zu Objekten
      - z.B. zur Bestimmung der Länge einer Liste gibt es die Built-In function len(list)
        - oop: list.length
      - Übersicht
        - https://www.w3schools.com/python/python_ref_functions.asp
  - Python stellt eine umfangreiche Klassenbibliothek zur Verfügung
    - Typische Funktionalitäten / Algorithmen werden als Methoden von Klassen bereitgestellt
      - z.B. String https://www.w3schools.com/python/ref_string_capitalize.asp

# Todo

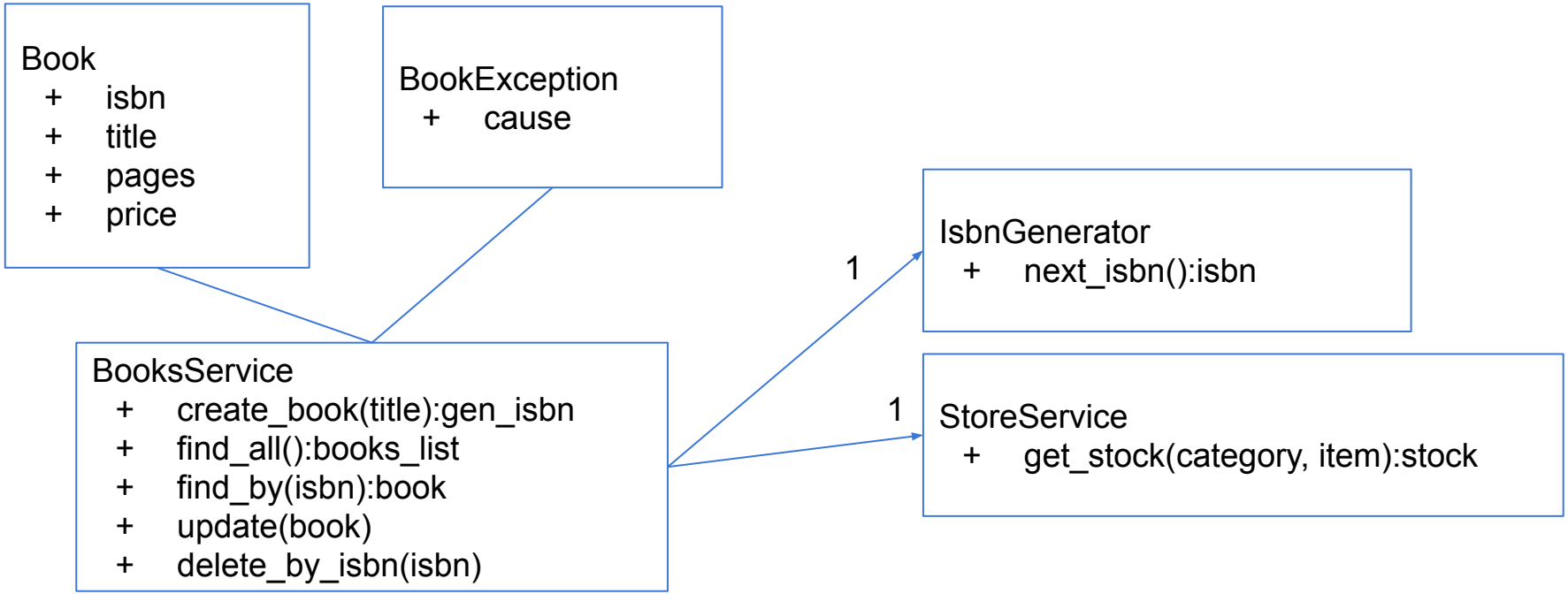


- Stellen Sie die marry / divorce auf einen dynamische Stil um
  - hasattr, delattr
- Verschaffen Sie sich einen Überblick
  - built-ins
  - String-Methoden
- range()-Funktion zum Erzeugen eines Range-Objektes z.B. zum Iterieren über einen Wertebereich
  - for (i= 0; i < 10; i = i + 1) -> Formulierung mit range
-

# Anwendungsprogrammierung

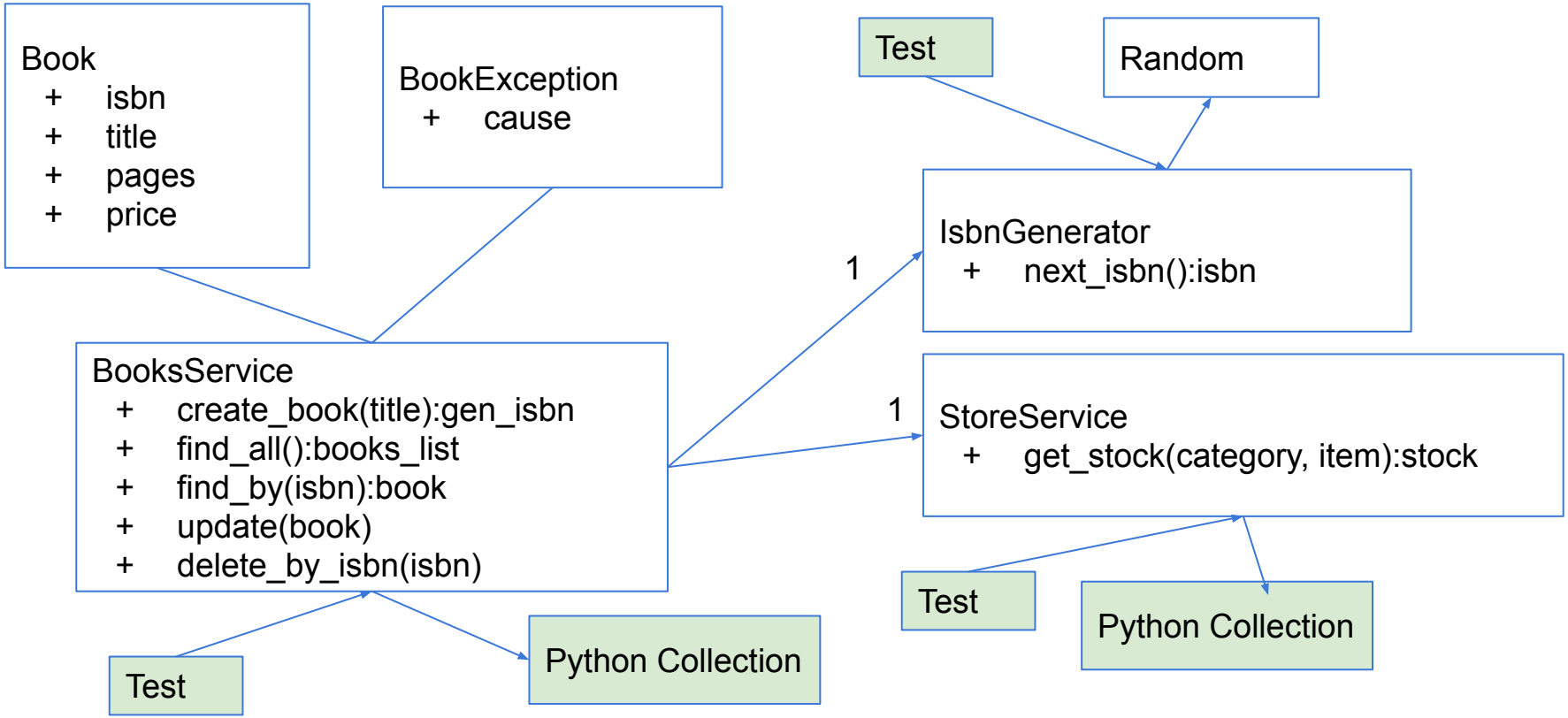
# Trainings-Anwendung





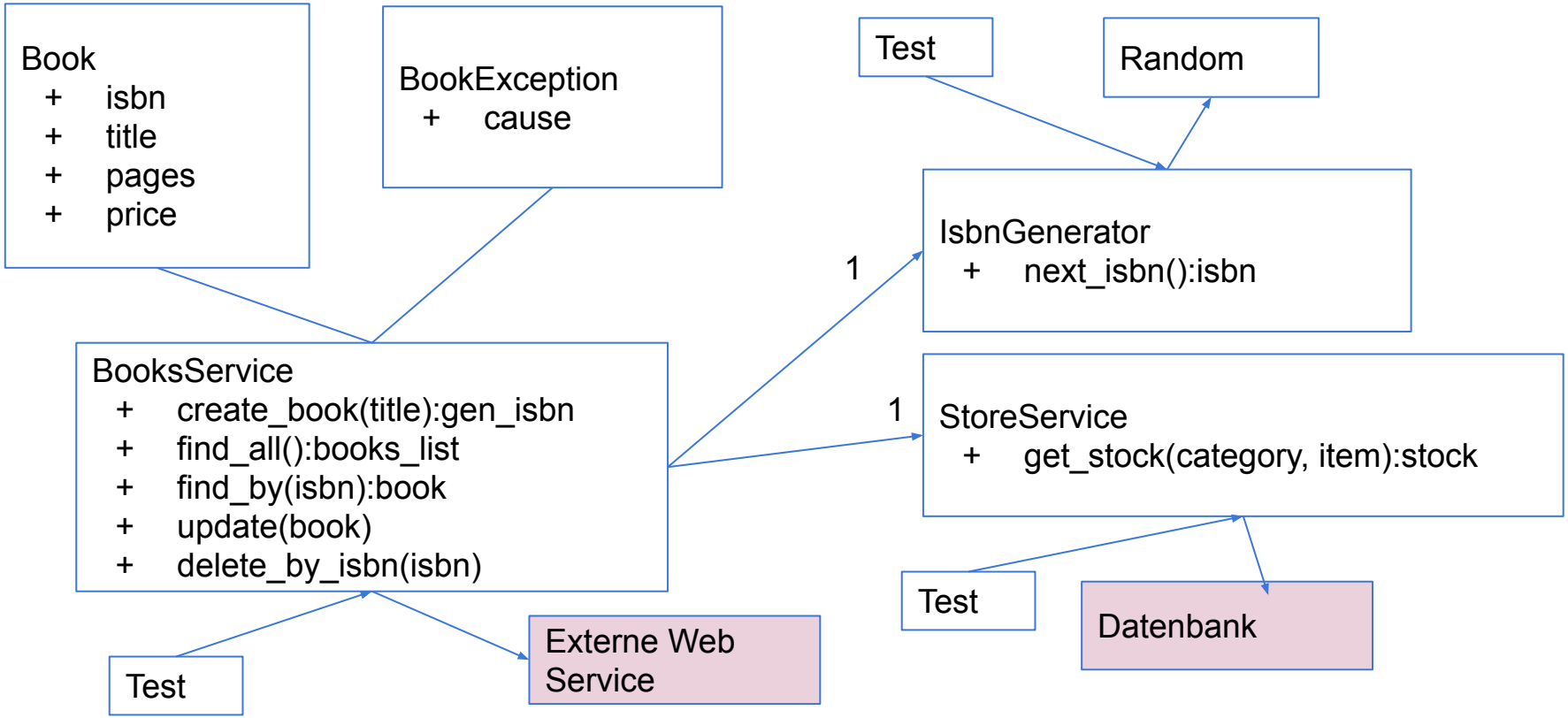
# Trainings-Anwendung: Resultat heute





# Trainings-Anwendung: Resultat Tag 3





# ToDo



- Übernehmen Sie das books-Verzeichnis von GitHub
- Kopieren Sie books in Ihr Projekt
- Passen Sie den Modul-Namen an
  - javacream -> Name Ihrer Organisation
- Verifizieren Sie
  - application.py ist lauffähig
  - Erzeugen Sie darin einen IsbnGenerator und generieren eine Isbn

# ToDo



- Implementieren Sie den StoreService auf Grundlage eines Dictionaries!
  - Nicht-vorhande Kategorie oder nicht vorhandenen Item -> return 0
- Zusätzliche Methoden
  - set_stock(category, item, stock)
  - get_categories():categorySet
  - get_items_for(category):itemSet
- Erweitern Sie die application.py
  - Hinweis
    - Im Wesentlichen führen Sie hier Tests ein
    - Etwas später -> UnitTest-Framework
- https://www.w3schools.com/python/python_dictionaries.asp

# Python Module

# (Fast) State of the Art



- Python-Module sind Artefakte in einem Artefakt-Repository
  - PyPi
  - Identifikation erfolgt über eine Koordinate
    - Name + Versionsnummer
- pip
  - Package Manager zum Installieren von Modulen aus dem Repository
  - Ablage der Bibliotheken erfolgt zentral im Verzeichnisbaum des Entwickler-Rechners
    - Projekt-abhängige Ablage kann nur durch eine Sandbox erfolgen, die eine komplette Python-Umgebung bereitstellt

# Beispiel: Modul unittest2



- Ein etabliertes Testing-Framework im Python-Umfeld
- pip install unittest2
- Testfälle üblicherweise gruppiert im test / tests -Directory
- Konventionen
  - name des Pythons: testxyz.py
  - Klasse erbt von Testcase
  - set_up(self)
  - test_abc(self)
    - Assertions
- Im test-Verzeichnis
  - python -m unittest2 discover .

# ToDo



- Testfälle für
  - StoreService
  - Optional: IsbnGenerator
- Definitiv nicht zu testen
  - Book
    - eine reine Datenstruktur
  - BookException
    - ?
  - BooksService
    - noch gar nicht implementiert…

# Exkurs: Context & Dependency Injection

# Übersicht



- Alle relevanten Fachobjekte, also bei uns BooksService, IsbnGenerator, StoreService werden von einem Context instanziert
- Der Context muss darüber hinaus die Dependencies setzen sowie die Objekte initialisieren
- CDI-Pattern ist damit eine Kombination, ein Meta-Pattern aus den GoF-Patterns Singleton, Factory und Strategy
-

# CDI in Python



- Es gibt kein komplett etabliertes CDI-Framework
  - Kein “Spring Python”
- Für unsere Anwendungen ist aber ein eigener Context total einfach zu implementierem

# Zu unserem BooksService

# Fachliche Spezifikation



- CRUD-Operationen
  - create(title):isbn | BookException
    - title muss mindestens aus einem Zeichen bestehen
    - price und pages können angegeben werden, falls nicht jeweils price=0, pages=1
    - isbn wird generiert
  - find_by_isbn(isbn) : book | BookException
    - isbn muss gesetzt sein
    - falls nicht oder kein Buch gefunden -> BookException
  - update(book): void | BookException
    - isbn muss vorhanden sein, title mindestens 1 zeichen, pages min 1, price >= 0
  - delete_by_isbn(isbn) : void | BookException
    - isbn muss gesetzt sein
    - falls nicht oder kein Buch gefunden -> BookException

# ToDo: Python als dynamische Sprache



- Das Decorator-Pattern am Beispiel einer Testutility, trace
- Idee: NotNoneChecker für Parameter

# ToDo Fachliche Erweiterung



- Zusätzliches Attribut im Book
  - available
- Bei einer Buch-Suche soll dieses Attribut durch eine store-Abfrage live bestimmt werden
-

# ToDo



- Umsetzung der Fachlichkeit
  - Im BooksService
  - In einem BooksServiceTest

# Python Module, Part 2: Eigene Module

# Was ist ein installiertes pip-Modul?



- Im Endeffekt ist jedes installierte Modul ein entpacktes Quellcode-Verzeichnis
- Root dieser Verzeichnisse ist im PYTHON_PATH zu finden
- Beim Laden = Importieren eines Moduls wird eine Initialisierung durchgeführt
  - __init__.py
    - Genauer: Durch eine __init__.py wird ein Verzeichnis zu einem Package
    - Ein Modul besteht aus einem oder mehreren Packages

# __init__py



- Initialisiert ein Package
- Stellt Meta-Informationen bereit
- __all__ = [‘name1’, ‘name2’,]
  - __all__ = [‘booksservice’, ‘context’, …]
- Bei definiertem __all__ funktioniert die Syntax
  - from module import *

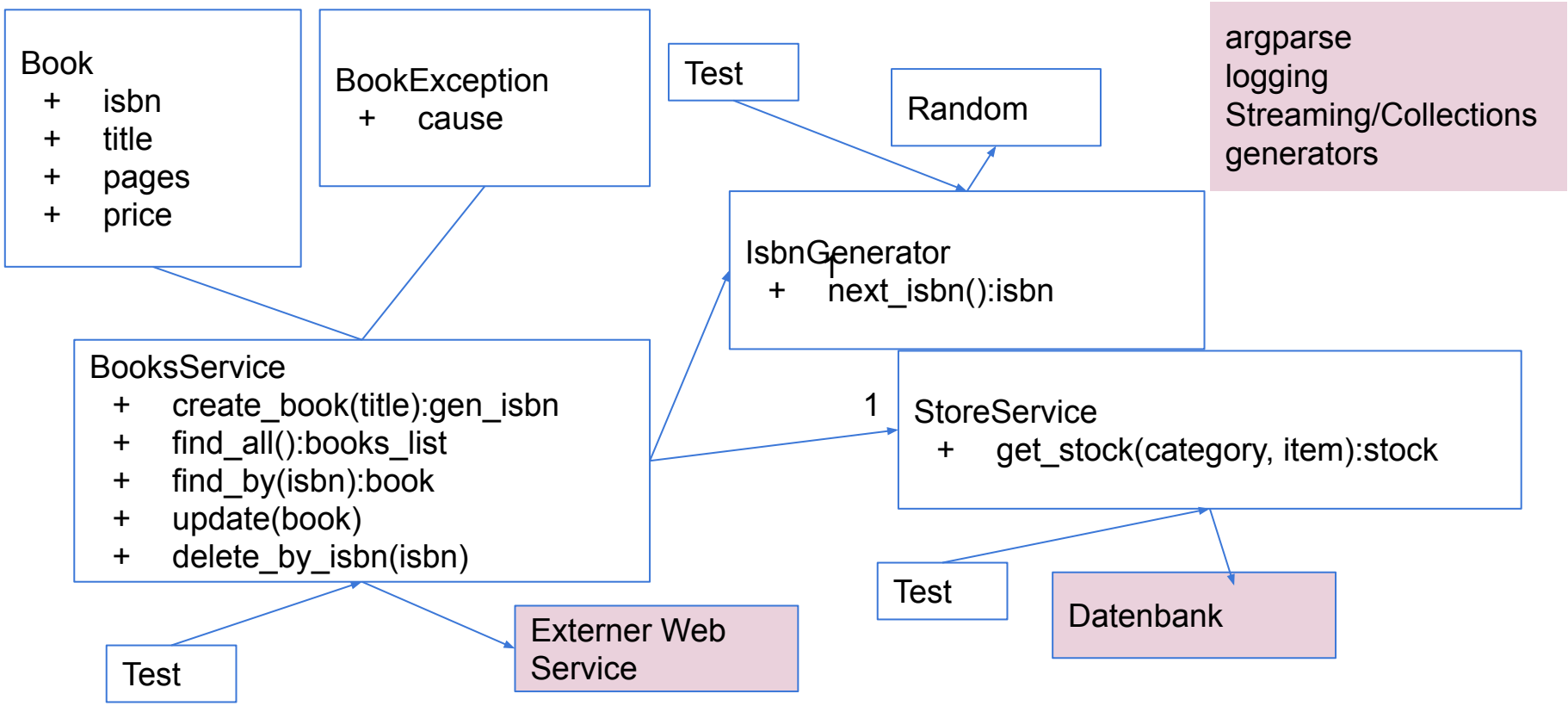
# Prinzipieller Ablauf



- Erstellen der setup.py
- python setup.py sdist
  - Erzeugt ein tarball-Archiv
- push auf Nexus
  - .pypirc
  - pip.conf
  - twine /dist/archive
- pull
  - pip install
  -

# Programm Tag 3





# Infrastruktur



- Ein RESTful WebService muss zugreifbar sein
  - http://h2908727.stratoserver.net:8080/swagger-ui.html
  - https://jsonplaceholder.typicode.com/
  - gerne auch jeder interne REST-Service
- MySql Datenbank
  - http://h2908727.stratoserver.net:8183/
    - phpadmin
      - user=user
      - Datenbank javacream
      - Port: 3406
  - gerne auch eine interne MySQL-Instanz

- Logging
  - Interne Python Library, “logging”
- MySQL
  - Notwendig ist der MySql-Connector bereitgestellt von der MySql-Community
  - Installation über pip install MySql-Connector
    - pip install mysql-connector-python
- REST-Aufrufe
  - Ein ganzer Satz von Bibliotheken ist verfügbar
  - Im Seminar: Requests
    - pip install requests
- Hinzufügen von Aufruf-Optionen zu einer ausführbaren Applikation
  - argparse
  - pip install argparse

# ToDo



- Alle Fachklassen sollen Debug-Meldungen schreiben, die alle Aufrufe protokolliert
  - Hinweis
    - Im test-Projekt hatten wir ja so was ähnliches…
    - Wie würden Sie das denn in der Anwendung umsetzen

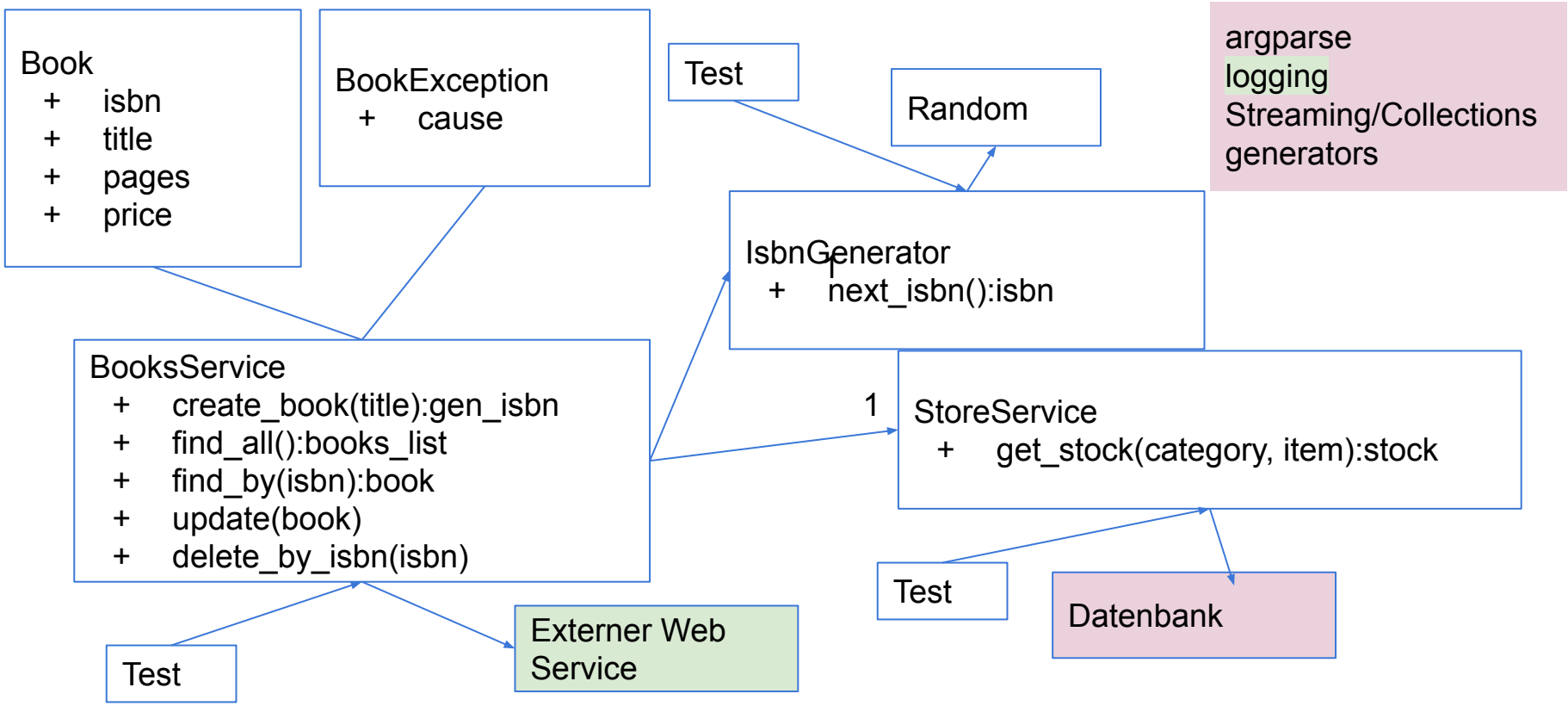
# ToDo



- Fix: Book-Klasse ohne pages!
- Implementieren Sie einen “BooksService” auf Basis von requests
- Einfach (nur Pfad notwendig
  - finder
  - create (ohne pages und price)
  - delete
- Komplizierter
  - Anlegen mit price-Header
  - update -> Book -> JSON -> Request-Body
- Optional
  - Generische Umwandlung Dict-Python-Objekt

# Aktueller Stand Tag 3





# Comprehensions



- Iteration, Transformation, Filtering in einer einzigen, kompakten Anweisung
  - [element * 2 for element in list if if%2 != 0]
- Hinweis
  - Was ist mit Lambda-Ausdrücken?
    - list.filter((element) => element %2 != 0).map((element) => element*2).collect(Collectors.toList())
  - Python: filter = lambda x :  x * 2
    - Einschränkung: Nur eine Anweisung

# ToDo



- Der WebService hat findAll(): List<Book>
  - Implementieren Sie diese Funktion im RestBooksService, BooksService
- Der Client soll nun noch ein paar Auswertungen auf der Book-Liste machen
  - Alle Bücher in einem Preisbereich
  - Alle verfügbaren Bücher
  - Alle Bücher, deren Titel…

# Dies und Das zu Collections



- Unpacking Operators
  - *
    - Liste
  - **
    - Dictionaries
- Generators
  - -scratch: Generators Demo

# ToDo



- DatabaseStoreService mit den Methoden get_stock und set_stock
- Vorsicht
  - Das immer wieder connecten auf die Datenbank ist sicherlich ineffizient
    - Connection Pooling?
- Überlegen Sie sich das Design