JAVACREAM

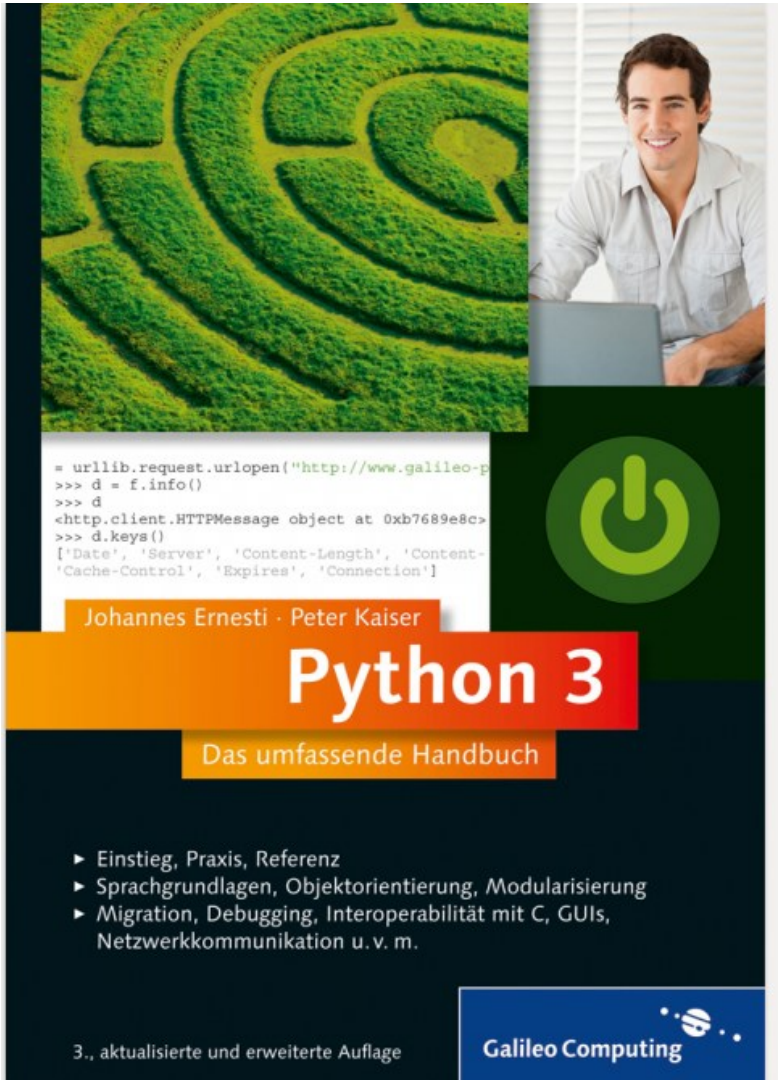
Training
Consulting
Projectmanagement

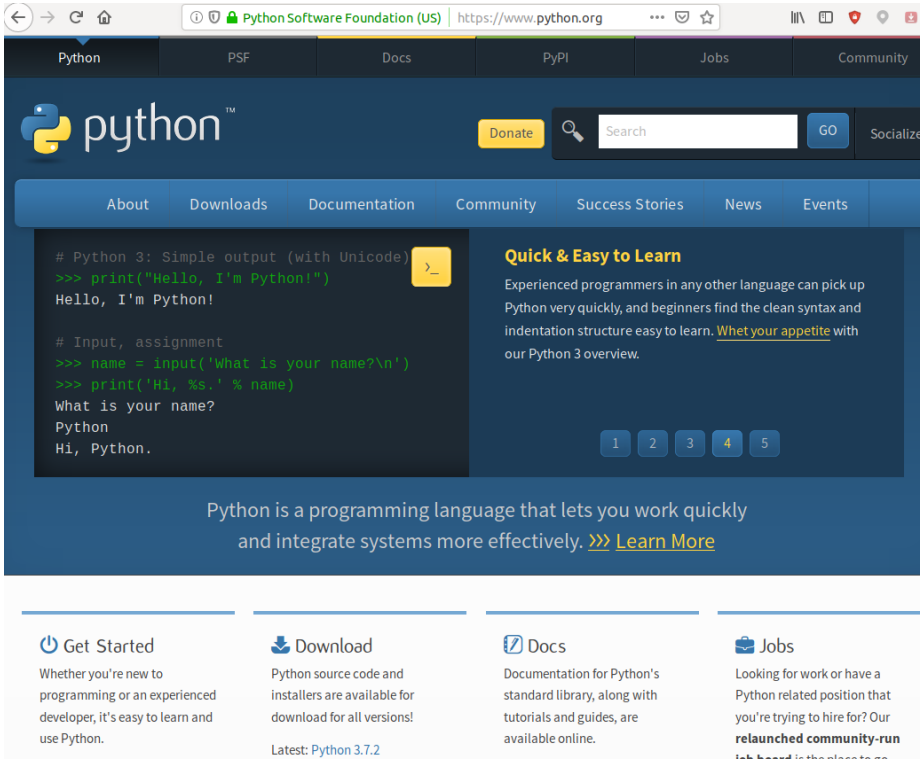
Python 3

Grundlagen der Programmierung

Literatur und Quellen

JAVACREAM
Training
Consulting
Projectmanagement





Einige Hinweise

JAVACREAM
Training
Consulting
Projectmanagement

- Die in diesem Seminar verwendete Werkzeuge und Frameworks sind Open Source
  - LPGL Lizenzmodell
- Dies ist ein Programmier-Seminar
  - Damit werden die Inhalte durch Übungen vertieft und verinnerlicht
- Dokumentation und Ressourcen stehen auch im Internet zur Verfügung

1

EINFÜHRUNG

1.1

VORSTELLUNG

Startpunkt

- Teilnehmende Person
  - Sicherer Umgang mit dem PC
    - Dateisystem
    - Browser
    - Konsole / Terminal
  - Recherche im Internet
    - Klassische Suchmaschinen
    - ChatGpt oder alternative Copilots
- Grundkenntnisse der Programmierung
  - Sind nicht erforderlich, aber hilfreich
  - "Ich habe bisher Tabellenkalkulation mit Excel gemacht. Genügt das?"
    - Ja

Hinweis

Eine Registrierung z.B. bei OpenAI etc. ist nicht notwendig

Ziel

- Am Ende des Seminars
  - Sind Sie leider keine Person, die sich sofort als "erfahrene ProgrammiererIn" bewerben kann
    - dazu fehlt jedoch "nur" praktische Erfahrung
  - Sie kennen etwa 90% der Python-Syntax
    - Decorators, Protokolle und Metaprogrammierung sind jedoch wirklich nicht unbedingt notwendig, um Anwendungen zu programmieren
  - Sie können jedoch anhand der praktischen Beispiele bereits anspruchsvolle Anwendungen programmieren

Ergänzungen

JAVACREAM
Training
Consulting
Projectmanagement

- Praktisch unvermeidbar sind Kenntnisse eines Source Code Management Systems (Versionsverwaltung)
  - Wir empfehlen Ihnen hierzu Git
- Weiterführende Python-Themen
  - Datenanalyse / Machine Learning mit Pandas, SciKit
  - Netzwerkprogrammierung z.B. mit Paramiko
  - Testskripte und –automatisierung
  - Web Anwendungen inklusive RESTful WebServices mit Django

Vorstellungsrunde

JAVACREAM
Training
Consulting
Projectmanagement

- Erzählen Sie etwas über sich
  - Name
  - Rolle im Unternehmen
  - Themenbezogene Vorkenntnisse
  - Aktuelle Problemstellung
  - Konkrete individuelle Zielsetzung

1.2

EIN PAAR GRUNDLEGENDE DINGE

Python – Eine Skriptsprache

- Kein aufwändiger Build-Prozess erforderlich
  - Die Skripte sind direkt ausführbar
- Portabilität
  - Die Python-Laufzeitumgebung steht für alle gängigen Programmiersprachen zur Verfügung
- Agilität der Software-Entwicklung
  - Änderungen des Skript-Codes werden von der Laufzeitumgebung sofort übernommen
- Die Struktur der Programme kann einfach gehalten werden
  - Anweisungen können ohne jeglichen Rahmen "einfach so" geschrieben werden
- Ausführung in einer REPL (Read–eval–print loop)

Hinweis

Diese Eigenschaften waren und sind kein Alleinstellungsmerkmal von Python!

1.3

DIE ARBEITSUMGEBUNG

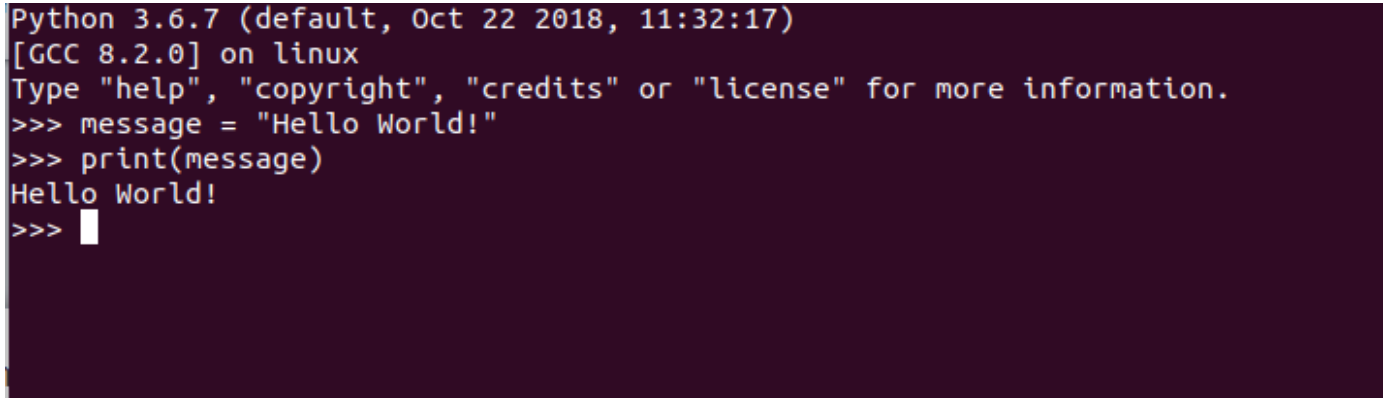
Installation

- Klassische (Antike?) Variante
  - Download und Installation der Python-Distribution
  - Download und Installation der Visual Studio Code Distribution
  - Installation der Python-Extension in Visual Studio Code
  - Einrichten von Virtual Environments pro Projekt
- Moderne Variante
  - Download und Installation der Visual Studio Code Distribution
  - Installation der Python-Extension in Visual Studio Code
  - Nutzen eines Development-Containers mit isolierter Python-Umgebung
- Im Training
  - Die klassische Variante ist umgesetzt
  - Wir benutzen keine Virtual Environments, wir haben nur ein Projekt "Training"

Admin-Rechte notwendig!

Details zur Python-Installation

- Python ist frei erhältlich
- Distributionen für alle relevanten Betriebssysteme verfügbar
  - Linux
  - Windows
  - Mac
  - …
- In der Distribution ist auch eine Dokumentation enthalten
  - Selbstverständlich auch Online verfügbar!
- Weiterhin die Python-Shell
  - Eine interaktive REPL



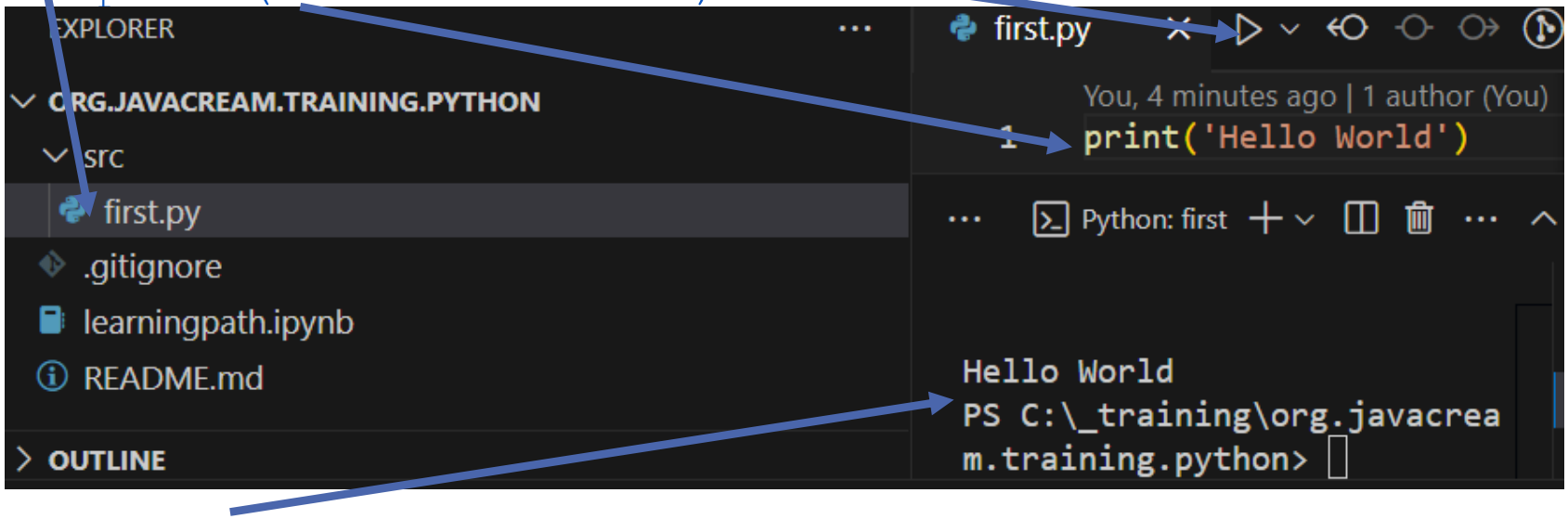
Anlegen des Python-Projekts in Visual Studio Code

JAVACREAM
Training
Consulting
Projectmanagement

- In Visual Studio Code wird ein Verzeichnis geöffnet
  - Name und Lokation dieses Verzeichnisses ist beliebig
- Im Anschluss daran
  - ist nichts mehr zu tun, das war es bereits...
- In einem realen Projekt würde nun noch eine Anbindung an Git durchgeführt werden
  - Falls Sie das tun wollen und Zugriff auf einen Git Server haben: Gerne!

Ein erstes Programm

- Erstellen Sie das "klassische" erste Programm in Python in der Datei `first.py` und Starten das Programm
- `print("Hello World!")`



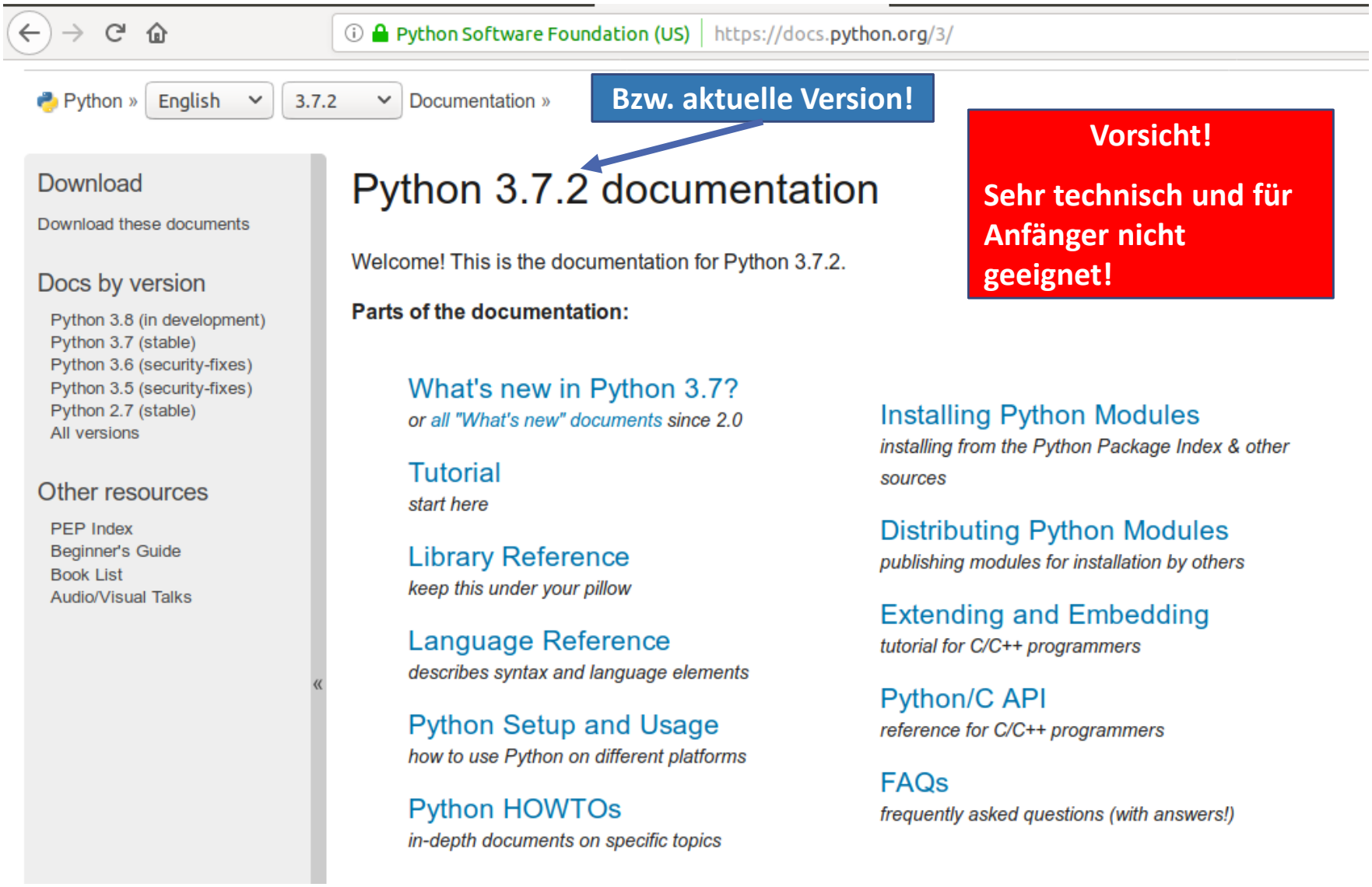
- Die Ausgabe erfolgt im integrierten Terminal von Visual Studio Code
- Es ist tatsächlich so einfach!

1.5

DOKUMENTATION UND RESSOURCEN

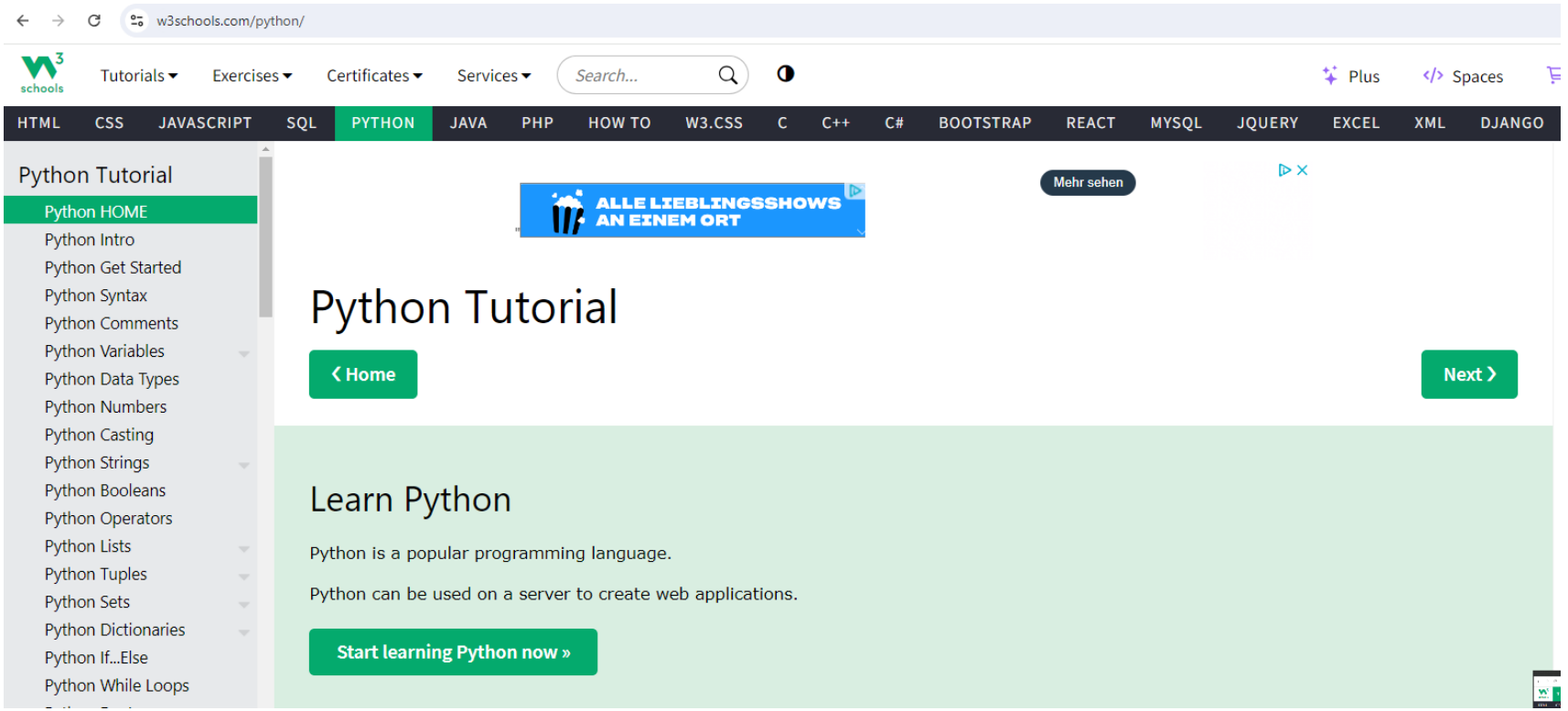
Python Online Dokumentation

JAVACREAM
Training
Consulting
Projectmanagement



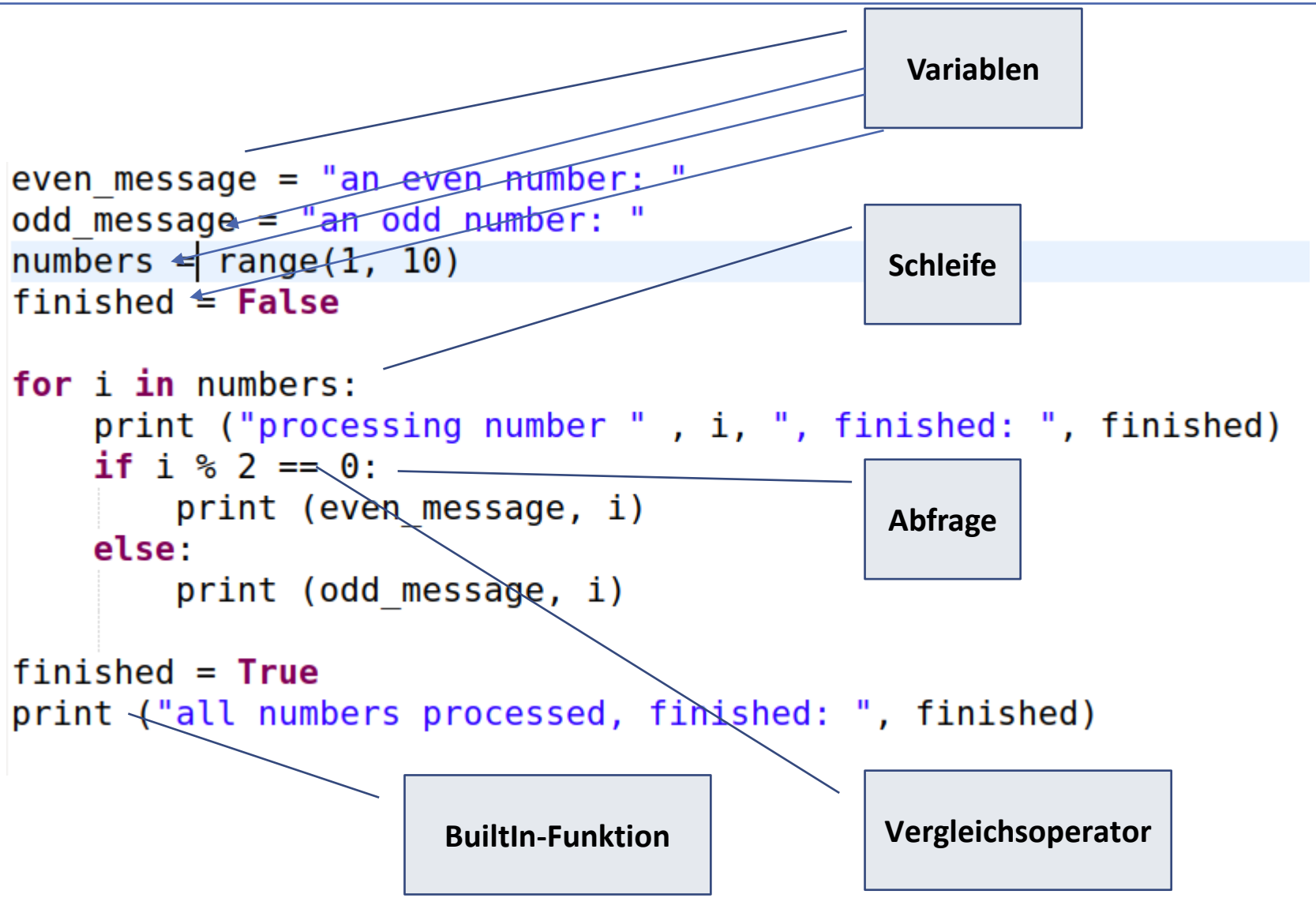
w3school – Eine öffentliche Lernplattform

JAVACREAM
Training
Consulting
Projectmanagement



Was erwartet uns im nächsten Teil?

JAVACREAM
Training
Consulting
Projectmanagement




```
even_message = "an even number: "
odd_message = "an odd number: "
numbers = range(1, 10)
finished = False

for i in numbers:
    print ("processing number " , i, ", finished: ", finished)
    if i % 2 == 0:
        print (even_message, i)
    else:
        print (odd_message, i)

finished = True
print ("all numbers processed, finished: ", finished)
```

2

VARIABLEN

2.1

ÜBERSICHT

Prinzip

- Eine Variable hat einen **Namen**
  - message
- sowie einen **Wert**
  - 'Hello'
    - Ein in einem Programm angegebener fixer Wert wird als **Literal** bezeichnet
- der mit dem '**=**'-Operator einer Variablen zugewiesen wird
  - message = 'Hello'

- Namen in Python in Englisch
  - Deutsche Bezeichner führen zu holprig lesbarem Code
    - länge = len('Hugo')
    - print(länge)
- Namen ausschließlich in Kleinbuchstaben
  - zur besseren Lesbarkeit je nach Betonung mit Unterstrichen arbeiten
    - statt `personlastname` besser `person_lastname`
  - Die Alternative mit eingestreuten Großbuchstaben (`personLastname`) wird in der Python-Community eher selten benutzt
- Namen bitte nur mit ASCII-Zeichen, obwohl der gesamte Unicode-Zeichensatz genutzt werden kann
  - Und ja: Sie könnten auch Emojis nutzen

2.2

ERSTES PROGRAMMIEREN MIT VARIABLEN

Zahlen

- Alles was mit einer Zahl beginnt, ist eine Zahl
- Die Genauigkeit ergibt sich aus dem Literal
  - Fließkommazahl, float
    - Mit Komma
    - 4.2
  - Ganzzahl, integer
    - Ohne Komma
    - 42

Neben der hier benutzten Dezimalschreibweise können Zahlen theoretisch auch oktal und hexadezimal angegeben werden

Strings

- Markierung durch einfache oder doppelte Anführungsstriche
  - Mischung ist nicht zulässig

- Die \ in Zeichenketten werden für spezielle Zeichen verwendet
  - z. B. \n für "Neue Zeile"
- Längere Stringkonstanten lassen sich über Tripelquotes erzeugen

```
""" Dies ist ein
    mehrzeiliger
    String"""
```

- **Formatierte Strings** haben Zugriff auf Variablen

```
number = 42
message = f'the number is {number}'
```

Konsolenausgabe und -eingabe

- Programme können interaktiv sein:
  - Benutzereingaben werden mit `input` angefordert
  - Ausgaben werden auf die Konsole mit `print` ausgegeben
- Dies sind sogenannte BuiltIn-Funktionen, die an jeder Stelle eines Python-Programms genutzt werden können

Funktionen werden weiter unten genau behandelt, hier pragmatisch:

- **Damit eine Funktion etwas tut, muss sie aufgerufen werden, dazu wird ein rundes Klammernpaar benutzt**
- **Benötigt die Funktion Informationen, also beispielsweise "was soll ausgegeben werden?", so wird dies als Literal oder Variable in die runden Klammern geschrieben**

Stand

- Deklaration von Variablen und Zuweisung eines Literals
  - Zahlen
  - Zeichenketten
- Nutzen der BuiltIn-Funktionen `input` und `print`
- Ein erstes interaktives Programm mit Konsoleneingabe und –ausgabe ist realisiert

- **Neben Zahlen und Zeichenketten unterstützt Python weitere Literale (logische Werte, Listen, ...), die weiter unten eingeführt werden**
- **In Python Version 3 gibt es etwa 70 BuiltIn-Funktionen**

3

DATENTYPEN UND OPERATOREN

3.1

LITERALE UND DATENTYPEN

Literale revisited

- Bisher
  - message = 'Hello'
  - Interpretation
    - "Der Variable message wird der Wert Hello zugewiesen"
- Nun
  - 'Hello' ist ein Zeichenketten-Literal, somit ist `message` eine Zeichenkette
    - Englisch: Zeichenkette = String
      - 'Hello' ist ein String-Literal, somit ist `message` ein `str`
  - Völlig äquivalent
    - 42 ist ein `int`-Literal
    - 4.2 ist ein `float`-Literal

Typisierung

- Der Typ des Literals wird implizit auch der Variablen zugewiesen
- Damit weiß Python zur Laufzeit immer ganz genau, welchen Typ eine Variable hat
  - Zur Bestimmung wird die BuildIn-Funktion `type` genutzt
    - Die Ausgabe ist etwas kryptisch, enthält aber genau das, was wir brauchen
      - `<class 'str'>`
      - `<class 'int'>`
      - `<class 'float'>`

- **Einer Variable kann jederzeit ein neuer Wert (Literal, andere Variable) zugeweisen werden, damit ändert sich potenziell auch der Typ der Variable**
  - **Python ist eine dynamisch typisierte Sprache, in einer statisch typisierten Sprache (Java, TypeScript, C#) muss bei einer Neuzuweisung der Typ passen**

Typumwandlung

- Der Typ einer Variablen kann, falls inhaltlich möglich, in einen anderen Typen umgewandelt werden
- Dazu wird für jeden Typen eine eigene BuiltIn-Funktion bereitgestellt
  - Der Name ist exakt derselbe, wie der relevante Teil der `type`-Ausgabe
- Umwandlung in Strings
  - `str(value)`
    - Geht immer
- Umwandlung in eine Zahl
  - `int(value)`
  - `float(value)`

- **Die Umwandlung in einer Zahl kann fehlschlagen**
  - **int('Hugo')**
  - **float('42komma3')**
  - **float('42,3')**

3.2

OPERATOREN

Was sind "Operatoren"?

JAVACREAM
Training
Consulting
Projectmanagement

- Ganz Allgemein
  - "Mit Operatoren werden in einer Programmiersprache Aktionen mit Variablen durchgeführt"
- Sind Operatoren damit nicht dasselbe wie BuiltIn-Funktionen?
  - Korrekt
- Warum dann der Unterschied?
  - Es geht eigentlich nur um die Lesbarkeit des Codes!
- Bitte etwas konkreter! Warum BuiltIns und Operatoren?
  - Wir wollen das Ergebnis der Addition der beiden Zahlen 20 und 22 bestimmen
    - Nutzen wir doch einfach den +-Operator
      - `result = 20 + 22`
    - Mit einer hypothetischen `plus`-BuiltIn-Funktion würde das so ausschauen
      - `result = plus(20, 22)`
  - Nun wollen wir das Ergebnis auf die Konsole ausgeben
    - Nutzen wir doch die `print`-Funktion
      - `print(result)`
    - Und was wäre hierfür ein griffiger Operator? Ein Emoji???
      - `result` 🖨️

Operatoren und Datentypen

- Operatoren sind nur zwischen kompatiblen Datentypen erlaubt
  - Beispiel
    - Die Summe aus zwei Zahlen kann berechnet werden, ebenso z.B. die Differenz
    - Was aber ist die Differenz zweier Zeichenketten? Das ist nicht definiert
  - Hinweis
    - Ob ein Operator für Datentypen unterstützt ist oder nicht entscheidet die Python-Community
      - Das ist damit etwas willkürlich
        - Multiplikation von zwei Zahlen mit * -> Ja
        - Multiplikation von zwei Strings -> Nein
        - Multiplikation eines Strings mit einer Ganzzahl -> Ja
        - Addition von zwei Strings mit + -> Ja
        - Addition eines Strings mit einer Zahl -> Nein
- Manche BuiltIn-Funktionen prüfen, ob die übergebenen Parameter einen gültigen Typ haben
  - `len('Hugo') # 4`
  - `len(42) # Fehler`

Operationen mit Zahlen

- Grundrechenarten
  - +
  - -
  - *
  - /
- Modulo und Potenzieren
  - %
  - **
- Reihenfolge
  - "Punkt vor Strich"
  - Die Reihenfolge kann durch runde Klammern definiert werden

Strings

- Werden zwei Zeichenketten addiert, werden sie zusammengefügt

```
'Spam' + 'Spam'
```

'SpamSpam'

- Zeichenketten können "multipliziert" werden (nur mit Zahlen):

```
'Spam'*3
```

```
'SpamSpamSpam'
```

- Die Funktion `len` bestimmt die Länge von Strings

```
len("Hallo")
  5
len("SpamSpam")
  8
```

3.3

TODO: BMI-BERECHNUNG

Zur Aufgabenstellung

- BMI = Body Mass Index, ein Wert zur Bestimmung des Fettgehalts eines Menschen, um daraus ein Kriterium zur Beurteilung von Krankheitsrisiken zu erhalten
  - https://de.wikipedia.org/wiki/Body-Mass-Index
- Name, Körpergröße und Gewicht werden vom Benutzer eingegeben

- **Schon mal ChatGpt oder eine Alternative zum Programmieren benutzt?**
  - **Vielleicht nicht zum kompletten Lösen der Übungen, aber zur Recherche, zum Spicken oder Ideen sammeln brilliant!**

4

KONTROLLSTRUKTUREN

4.1

VERGLEICHSOPERATOREN UND DER DATENTYP BOOL

Boolean Werte und Operatoren

JAVACREAM
Training
Consulting
Projectmanagement

- Es gibt exakt zwei logische Zustände: Wahr und Falsch
  - Keine Zwischenzustände wie "vielleicht" oder Wahrscheinlichkeiten wie 75% erlaubt, das ist Statistik
- Python kennt dafür die beiden Literale `True` und `False`
- Operatoren
  - `not`
    - Negation, wahr wird falsch und umgekehrt
  - `and`
    - logische Und-Verknüpfung, das Ergebnis ist nur dann wahr, wenn beide Werte wahr sind
  - `or`
    - logische Oder-Verknüpfung, das Ergebnis ist nur dann falsch, wenn beide Werte falsch sind

Vergleichsoperatoren

- Vergleich mit
  - `==`
    - Gleichheit der Werte/Inhalte
  - `!=`
  - `<`
  - `<=`
  - `>`
  - `>=`
  - `is`
    - Gleichheit Referenzen
- Ergebnis der Operatoren ist `True` oder `False`

4.2

BEDINGTE PROGRAMMAUSFÜHRUNG

Abfragen

- Ablaufsteuerung im `if-else-elif`

```
if a==3:
   print "a ist 3"
elif a==4:
   print "a ist 4"
else:
   print "weder 3 noch 4"
```

- `elif` und `else` sind optional
- Die Einrückung bestimmt den Block
  - Klammern sind nicht nötig

- **Einrückungen sind in anderen Sprachen optional, werden jedoch häufig durch Code-Formatierungsregeln im Endeffekt doch genutzt**

Pattern Matching in Python

- Neu eingeführt in Python 3.10
- Eine mächtige Methode, um Werte und Strukturen zu prüfen und zu verarbeiten
- Beispiel

```
command = input('please enter a command: ')
match command:
        case "start":
            print("start")
        case "stop":
            print("stop")
        case _:
            print(f"unknown command {command}")
```

- **Ähnlich wie switch-Anweisungen in anderen Sprachen, aber wesentlich flexibler**

4.3

SCHLEIFEN

Schleifen

- Zwei Arten von Schleifen:
  - `while`
  - `for`
    - wird weiter unten bei den Datencontainern eingeführt
- `while`

```
counter = 0
while counter < 3:
  print (counter)
  counter = counter + 1
```

Break und Continue

JAVACREAM
Training
Consulting
Projectmanagement

- `break`
  - beendet die aktuelle Schleife
- `continue`
  - bricht den aktuellen Schleifendurchlauf ab, macht mit dem nächsten weiter

4.5

TODO: BMI-KATEGORISIERUNG

Zur Aufgabenstellung

- Nach Standard-Kriterien erfolgt die Beurteilung in 4 Kategorien von 'untergewichtig' bis 'fettleibig'
  - https://de.wikipedia.org/wiki/Body-Mass-Index

5

DATENCONTAINER

5.1

MOTIVATION

Datencontainer: Motivation

- Mit einem Container können zusammengehörige Informationen gruppiert werden
- Beispiel
  - Bisher
    - name = 'Sawitzki'
    - height = 183
  - Idee
    - p1_name = 'Sawitzki'
    - p1_height = 183
  - Liste mit Personen-Informationen
    - p1 = ['Sawitzki', 183]
    - p2 = ['Musterperson', 176]

es ist nicht eindeutig klar, dass `height` und `name` irgendwie zusammengehören

Besser, aber nur eine Namenskonvention

Eine Liste gruppiert alle zusammengehörenden Informationen
Nochmal besser, aber weiter unten wird es richtig gut!

5.2

KATEGORIEN

Liste in Python

- Eine Aufzählung von beliebig vielen Elementen
- Literal: `[...]`
  - `my_list = [e1, e2, e3]`
- Type: `list`
- Elementzugriff
  - Angabe eines numerischen Index als Ganzzahl in eckigen Klammern
    - beginnend bei 0

Benennen Sie Variablen nie wie einen Typen!

Sie überschreiben damit das Standard-Verhalten und das ist in den allermeisten Fällen ein unbeabsichtigter Effekt!

- **Einen Indexzugriff ausserhalb des gültigen Bereiches (0 bis zur Länge – 1) verursacht einen Fehler**
- **Auch negative Indizes sind möglich, -1 entspricht dem letzten Element**

Dictionary in Python

- Eine Aufzählung von beliebig vielen Key=Value-Paaren
- Literal: `{...}`
  - `my_dict = {k1: v1, k2: v2, k3: v3}`
- Type: `dict`
- Elementzugriff
  - Angabe des Keys in eckigen Klammern

- **Ein Zugriff mit vorhandenem Key verursacht einen Fehler**
- **Dictionaries werden in anderen Programmiersprachen auch als Map bezeichnet**

Set / Menge in Python

- Eine Aufzählung von beliebig vielen Elementen mit Duplikatserkennung
- Literal `{...}`
  - `my_set = {e1, e2, e3}`
- Type: `set`
- Ein Set hat keinen Element-Zugriff
  - und damit keine innere Ordnung

- **Ein leeres geschweiftes Klammernpaar erzeugt ein Dictionary**
  - **Wozu auch leere Collections genutzt werden können sehen wir weiter unten**

Allgemeine Eigenschaften der Collections

- Jede Collection hat eine Länge
  - Bestimmung mit der BuiltIn-Funktion `len`
- Über alle Collections kann iteriert werden
  - `for element in collection:`
    - …
- Collections können auch über eine BuiltIn-Funktion erzeugt werden
  - Der Name der BuiltIn-Funktion entspricht dem Typen
    - `my_list = list()` erzeugt, wie `[]` eine leere Liste
  - Diese Funktionen können auch zur Typumwandlung genutzt werden
- mit in kann geprüft werden, ob ein Element in einer Liste / Set oder ein Key in einem Dictionary enthalten ist
  - `if element in collection:`

Range in Python

- Eine fortlaufende Liste von Ganzzahlen
- Erzeugung nur mit BuiltIn-Funktion
  - `my_range = range(1, 5)`
    - start inklusive, ende exklusiv
  - `my_range = range(1,5,2)`
    - 2: step
- Type: `range`
- Elementzugriff
  - Bei der Liste Angabe eines numerischen Index, eine Ganzzahl
    - beginnend bei 0
  - Allerdings werden Ranges häufig für eine Iteration über ein Zahlenintervall genutzt

Tupels in Python

- Eine Aufzählung einer **fixen Anzahl** von Elementen
- Typ: `tuple`
- Erzeugung mit `tuple(['Summer', 'Autumn', 'Winter', 'Spring'])`
- Elementzugriff
  - Beim Tupel Angabe eines numerischen Index, eine Ganzzahl
    - beginnend bei 0

- **Tupels sind im Gegensatz zu Listen unveränderbar**
  - **das sehen wir weiter unten**

6

EIN KOMPLEXERES BEISPIEL

6.1

AUFGABENSTELLUNG

Vorgabe

- Als Datenbasis dient der folgende Auszug
  - '81371': 'München',
  - '70567': 'Stuttgart',
  - '30000': 'Berlin',
  - '81333': 'München',
  - '30001': 'Berlin',
  - '30002': 'Berlin',
  - '40005': 'Hamburg'
- Es sollen zwei Anwendungen erstellt werden
  - Welche Stadt gehört zu einer Postleitzahl?
  - Welche Postleitzahlen gehören zu einer Stadt?

- **Die Eingaben erfolgen über die Konsole**
- **Die Konsolenausgaben zeigen die gefundenen Treffer bzw. eine sprechende Information**

7

FUNKTIONEN

7.1

ÜBERSICHT

Funktionen

- Kleinste "Einheit" von wiederverwendbaren Code
- Definition mit `def`

```
def my_function():
  print "called my_functionut"
```

- Funktionen werden aufgerufen durch runde Klammern `()`

```
my_function()
```

Deklaration

JAVACREAM
Training
Consulting
Projectmanagement

```
def calculate_price(original , discount, shipping):
    discounted_price = original * (1 - discount/100)
    price = discounted_price + shipping
    return price

def calculate_shipping (provider):
    if provider == "EIM":
        shipping = 5.99
    elif provider == "CGK":
        shipping = 3.99
    else:
        shipping = 9.99
    return shipping

def order():
    provider = "EIM"
    shipping = calculate_shipping(provider)
    original_price = 19.99
    price = calculate_price(original_price, 10, shipping)
    print("total price: ", price)
```

Name der Funktion

Parameter

Rückgabe

Parameter

- Die Parameter der Funktion werden innerhalb der runden Klammern angegeben

```
def my_function(p):
  print(f"param: {p}"

my_function("Spam") # param: Spam
```

- Die Anzahl der übergebenen Parameter muss mit der Parameterliste übereinstimmen

- **Default-Werte für Parameter sowie variable Parameterlisten werden unten behandelt**

Gültigkeitsbereiche der Variablen

- In Funktionen definierte Variablen sind lokal, d.h. nur innerhalb der Funktion gültig
  - Im Unterschied zu einer Variablen, die in einer Funktion definiert wird, muss der Parameter beim Aufruf gesetzt werden

- **Variablen, die außerhalb einer Funktion definiert werden, können als "global" aufgefasst werden**
  - **Ein lesender Zugriff ist überall möglich**
  - **Ein schreibender Zugriff ändert jedoch den Wert der globalen Variable nicht (!), es wird im Endeffekt eine lokale Variable gleichen Namens angelegt**
    - **Soll auf Variablen außerhalb der Funktion (schreibend) zugegriffen werden, so müssen diese als `global` deklariert werden**

Zugegebenermaßen etwas verwirrend, aber zum Glück nicht zwingend zu nutzen...

Rückgabewerte von Funktionen

- Funktionen können Werte zurückliefern
  - Schlüsselwort `return`
- Dieser Rückgabewert kann wiederum in Zuweisungen oder weiteren Funktionsaufrufen verwendet werden

```
def mult(a,b):
  return a*b
print mult(4,2)
  8
```

7.2

EIN PRAKTISCHES BEISPIEL

8.1

PARAMETER

Optionale Parameter

- Bei der Funktionsdefinition können Parameter mit Default-Werten versehen werden. Beim Aufruf müssen für diese dann keine Werte angegeben werden

```
def f(a=5,b=2):
  print  a*b
f()
  10
```

Parameter-Liste

- Funktionsparameter die bei Definition mit einem * versehen werden, werden als beliebig langes Tupel übergeben

```
def f(*a):
  print a
f(1,2,3,4)
  (1, 2, 3, 4)
f("bla","bla")
  ('bla', 'bla')
```

Benannte Parameter

- Parameter die bei Definition mit ** markiert sind, werden im Endeffekt als Dictionary übergeben

```
def f(**a):
  print (a)


f(number=42,name="Hugo")
{'number': 2, 'name': 'Hugo'}
```

9

DATEIOPERATIONEN

9.1

LESEN UND SCHREIBEN VON DATEIEN

9.2

OPEN

Dateien öffnen und lesen

- Die Funktion open() öffnet eine Datei
  - `file = open('datei.txt', 'r')`
    - Der Modus 'r' steht für lesen
- read(): Liest den gesamten Inhalt der Datei
  - `content = file.read()`
  - `print(content)`
- readline(): Liest eine Zeile
  - `line = file.readline()`
- readlines(): Liest alle Zeilen und gibt eine Liste zurück
  - `lines = file.readlines()`
- Schließen der Datei
  - `file.close()`

Fehlerbehandlung

- Potenzielle Fehler beim Dateioperationen, wie z.B. Datei nicht gefunden werden durch spezielle Fehlertypen signalisiert
  - z.B. `FileNotFoundError`

```
try:
    with open('datei.txt', 'r') as file:
        content = file.read()
except FileNotFoundError:
    print("Datei nicht gefunden.")
```

Dateien schreiben

- Die Funktion open() öffnet eine Datei
  - `file = open('datei.txt', 'w')`
    - Der Modus 'w' steht für überschreiben
    - Der Modus 'a' steht für ergänzen
- write()
  - `file.write('Hello')`
- writelines(): Liest alle Zeilen und gibt eine Liste zurück
  - `lines = ['Hello', World']`
  - `file.writelines(lines)`
- Schließen der Datei
  - `file.close()`

9.3

CONTEXTMANAGER

Motivation

- Was sind Contextmanager?
  - Ein Werkzeug, um Ressourcen sicher zu verwalten
  - Automatisieren das Einrichten und Aufräumen von Ressourcen
    - Dateien
    - Netzwerkverbindungen
- Typische Anwendung
  - Öffnen und Schließen von Dateien
  - Sperren von Ressourcen
  - Transaktionen in Datenbanken
- Dazu wird die with-Anweisung genutzt
  - with open('datei.txt', 'r') as file:
  - content = file.read()

Lesen mit with


```
with open('datei.txt', 'r') as file:
    content = file.read()
```

- Die Datei wird hier automatisch geschlossen
  - Auch im Fehlerfall

Schreiben mit with


```
with open('datei.txt', 'w') as file:
    file.write('Hello')
```

- Die Datei wird hier automatisch geschlossen
  - Auch im Fehlerfall

10

MODULE

10.1

ÜBERSICHT

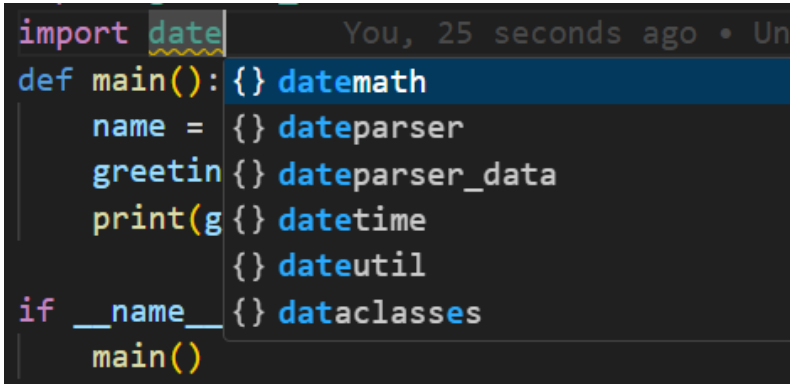
Entwicklung

- Bisher
  - BuiltIn-Funktionen
    - Stehen überall zur Verfügung
      - print('Hello')
  - Methoden sind Funktionen, die einem Objekt zugeordnet sind
- Neu
  - Wir können auch eigene Funktionen schreiben, die dann aufgerufen werden können
    - Stehen direkt nur in dem Python-Script zur Verfügung, in dem sie geschrieben wurden
  - Jedes Python-Script definiert ein eigenes Modul
    - Ein Modul ist ein Objekt, die darin enthaltenen Definitionen sind Eigenschaften dieses Objekts
  - Mit import wird ein Modul über eine Pfadangabe bereitgestellt

- Ein Modul stellt Funktionen und Variablen zur Verfügung
  - Keine lauffähige Anwendung!
    - Falls ein Modul gestartet wird, passiert nichts
  - In der Softwarearchitektur werden Module auch gerne als "Services" bezeichnet
- Eine Applikation ist die lauffähige Anwendung
  - diese importiert alle notwendigen Services
  - Die Applikation enthält eine main-Funktion
  - Der Aufruf der main-Funktion innerhalb der Applikation wird abgefragt
    - if __name__ == '__main__':
      - Damit kann auch ein Modul z.B. eine Testsequenz enthalten

Details zum import-Statements

- import <path_to_module>
  - Ein Pfad beginnt in dem Verzeichnis, in dem die Python-Applikation gestartet wird
    - Genauer: Im Systempfad der Python-Installation
  - Module, die in der Python-Installation vorhanden sind, werden ebenfalls gefunden
    - Hier erfolgt keine Pfadangabe, es genügt der Name des Moduls



Details zum import-Statements

JAVACREAM
Training
Consulting
Projectmanagement

- Der Objektname des Moduls ist im Standardfall der Name des Moduls
  - eventuell auch mit Pfad und damit lang
  - Änderung des Objektnamens mit as
    - import greeter_service as gs
- Direktes importieren der Modul-Eigenschaften
  - from greeter_service import greet

Modulkonzept in Python

- BuiltIn-Funktionen entstammen einem Python-Modul, das automatisch importiert wird
- Die Funktionalitäten einer modernen Programmiersprache sind enorm umfangreich
  - Zur Übersichtlichkeit werden diese Funktionen allesamt in Modulen gruppiert
  - Die Entscheidung, was alles im Standard-Modul vorhanden ist, ist subjektiv
    - str ->Ja, list -> Ja, Datumswert -> Nein
    - Dateien lesen, schreiben -> Ja, Auflisten eines Verzeichnisses -> Nein
    - Elementare mathematische Berechnungen -> Ja, Zufallszahl -> Nein

Eigenene Module

- Jedes Python-Programm ist bereits ein Modul
  - Name = Name der Datei ohne Dateiendung .py
    - Vorsicht: Dies gilt nur, wenn der Dateinamen auch syntaktisch als Modulname gültig ist!
- Sinn
  - Gekapselte Einheiten
  - Wiederverwendung von Code
  - Libraries

- **Hinweise**
  - **Python unterstützt für eigene Module auch sogenannte "packages"**
    - **Diese können eine Initialisierungsroutine enthalten**
    - **und durch Metainformationen nur definierte Elemente exportieren**
      - **nicht exportierte Elemente stehen nur innerhalb eines Packages zur Verfügung**
  - **Ein Beispiel hierfür ist weiter unten gegeben**

10.2

BEISPIELE

Das Modul math

- Konstanten
  - Pi, e, …
- Mathematische Berechnungen
  - Runden, Maximal- und Minimalwerte
  - Trigonometrische Berechnungen
  - …

Das Modul random

- `random()`
  - Zufallswert zwischen 0 und 1
- `randrange()`
- Zufallswert aus gegebenem Bereich
- `randint(min,max)`
- Ähnlich *randrange(),* jedoch ist hier die Obergrenze enthalten
- `sample(liste,anzahl)`
  - sucht eine Anzahl von Elementen zufällig aus der gegebenen Liste aus
  - Ergebnis ist eine Liste
- `choice(liste)`
  - sucht sich ein Element zufällig aus der Liste aus

Variablen von os


```
import os

os.name     # OS basistyp (NT,mac,posix, riscos)
os.platform# Platform (win32,linux,solaris,…)
os.sep      # \ für win; / fuer unix
os.linesep  # \r\n für win; \n für unix; \r fuer mac
```

Methoden von os


```
import os

os.path.isfile  # prüft, ob Parameter Datei ist
os.path.isdir   # prüft, ob Parameter Verzeichnis ist
os.path.getsize# Dateigröße
os.listdir      # Einträge im Verzeichnis
os.chdir        # wechselt das aktuelle Verzeichnis
os.getcwd       # liefert das aktuelle Verzeichnis
os.realpath     # löst relative Verzeichnisnamen auf
```

Das Modul datetime

- `time()`
  - Zeit in Sekunden seit dem 1.1.1970 0:00
- `clock()`
  - CPU-Zeit für diesen Prozess
- `sleep(n)`
  - Pause von `n` Sekunden
- `gmtime(t)`
  - nimmt Sekunden seit 1.1.1970 als Parameter und generiert daraus ein 9er Tupel mit den Informationen
    - Jahr (4stellig)
    - Monat
    - Tag im Monat
    - Stunde
    - Minute
    - Sekunde
    - Wochentag (Montag ist 0)
    - Tag im Jahr
    - DST (Daylight Saving)

Datumskonvertierung mit time

- `strftime()`
  - gibt die Zeit gemäß dem gegebenen Formatstring formatiert aus:

```
time.strftime("Uhrzeit: %H:%M:%S %d.%m.%Y")
```

- `strptime(String)`
  - Parst einen gegebenen String nach gegebenem Format

- **JSON** (JavaScript Object Notation) ist ein weit verbreitetes Format für den Datenaustausch, das menschenlesbar ist und von praktisch allen Programmiersprachen unterstützt wird
  - Das Standard-Format zum Datenaustausch im Internet
- Beispiel

```
import json

data = {'key': 'value'}
with open('data.json', 'w') as file:
    json.dump(data, file)

with open('data.json', 'r') as file:
    loaded_data = json.load(file)
```

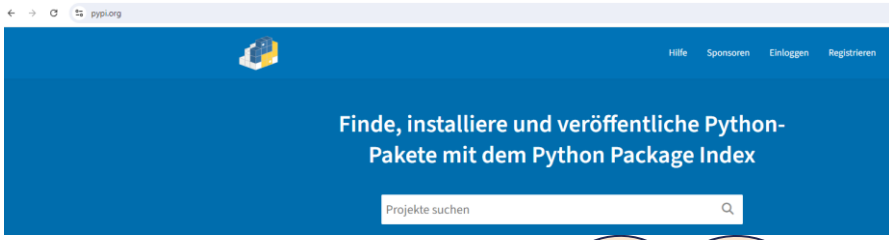
11

PIP UND PYPI

11.1

BEREITSTELLUNG VON MODULEN

Das Modul-Repository PyPi



Developer

Python-Installation

pip install paramiko

Module-Cache

PyPi-Repository
- **paramiko**
- **mysql-connector-python**
  - **v5.0**
  - **v8.0**

Externe Module

JAVACREAM
Training
Consulting
Projectmanagement

- Konsequenzen
  - Komplexere Projektorganisation
  - Abhängigkeiten zu externen Projekten mit unbekannten Release-Zyklen
  - Bug-Fixes erfordern erhöhten Kommunikationsaufwand
- Lösungen
  - Einsatz eines Build-Werkzeugs mit
    - Modul-Repository
    - Dependency Management
  - PyPI, Python Package Index

Einführung in PIP

- PIP steht für "Pip Installs Packages" und ist das Paketverwaltungssystem für Python
- Es ermöglicht die Installation und Verwaltung von Softwarepaketen, die in Python geschrieben sind
- Einführung in Python 2.7.9 und Python 3.4
  - Heute weit verbreitet in der Python-Community.

Verwendung von PIP

- installation
  - PIP wird in der Regel mit Python installiert
  - Prüfen mit pip –version
- Grundlegende Befehle
  - installieren eines Pakets
    - pip install paketname
  - Aktualisieren eines Pakets
    - pip install --upgrade paketname
  - Deinstallieren eines Pakets
    - pip uninstall paketname
  - Auflisten installierter Pakete
    - pip list

12

DIE OBJEKTORIENTIERTE SYNTAX

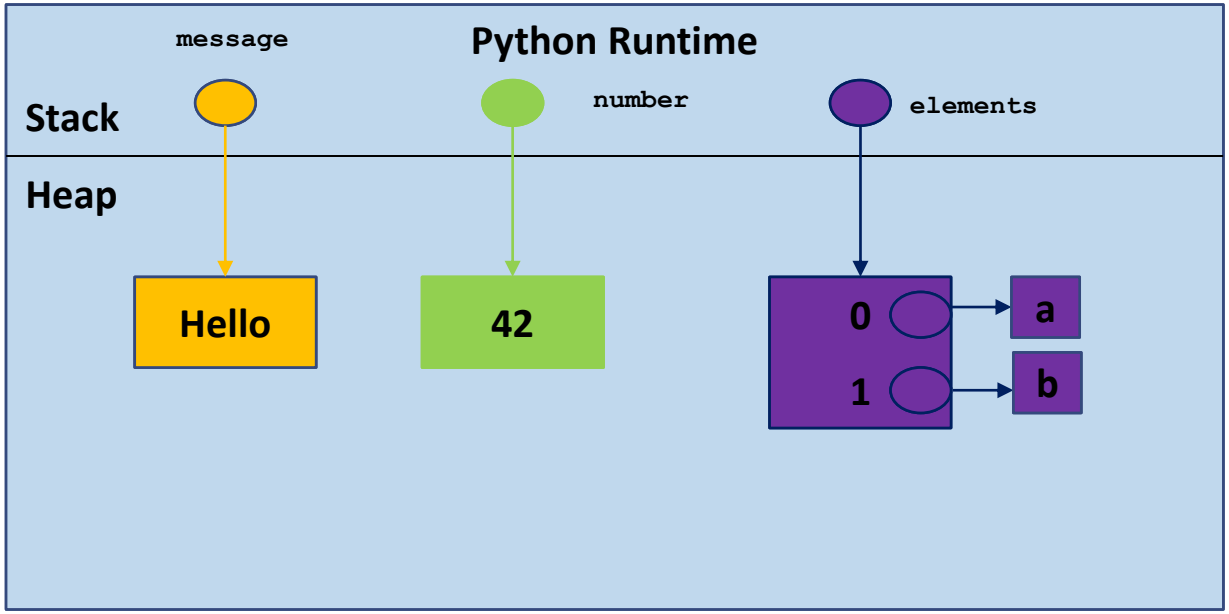
12.1

METHODEN

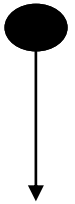
12.2

REFERENZEN

Speichermodell der Python Laufzeit



"Referenz"



```
message = 'Hello'
```

```
number = 42
```

```
elements = ['a', 'b']
```

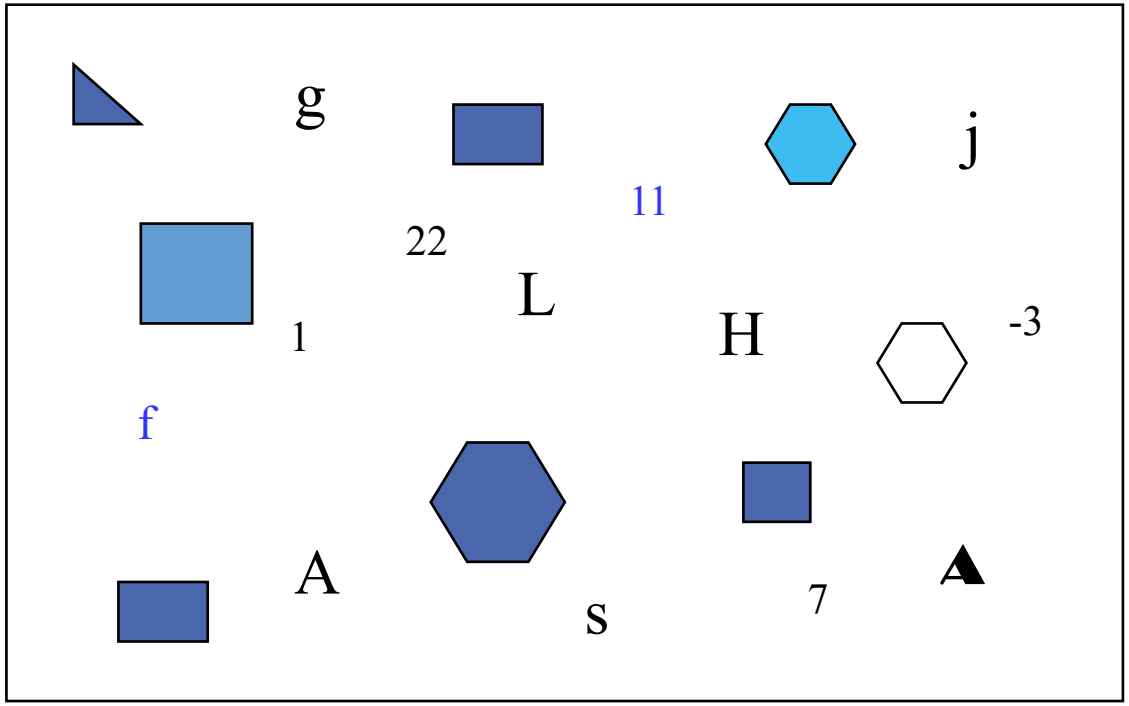
13

OBJEKTORIENTIERTE PROGRAMMIERUNG

13.1

DER OOP-ANSATZ

- Komplexes, ungeordnetes System, Zusammenhänge?
  - Was ist wesentlich, was unwesentlich?
  - Existieren Abhängigkeiten?

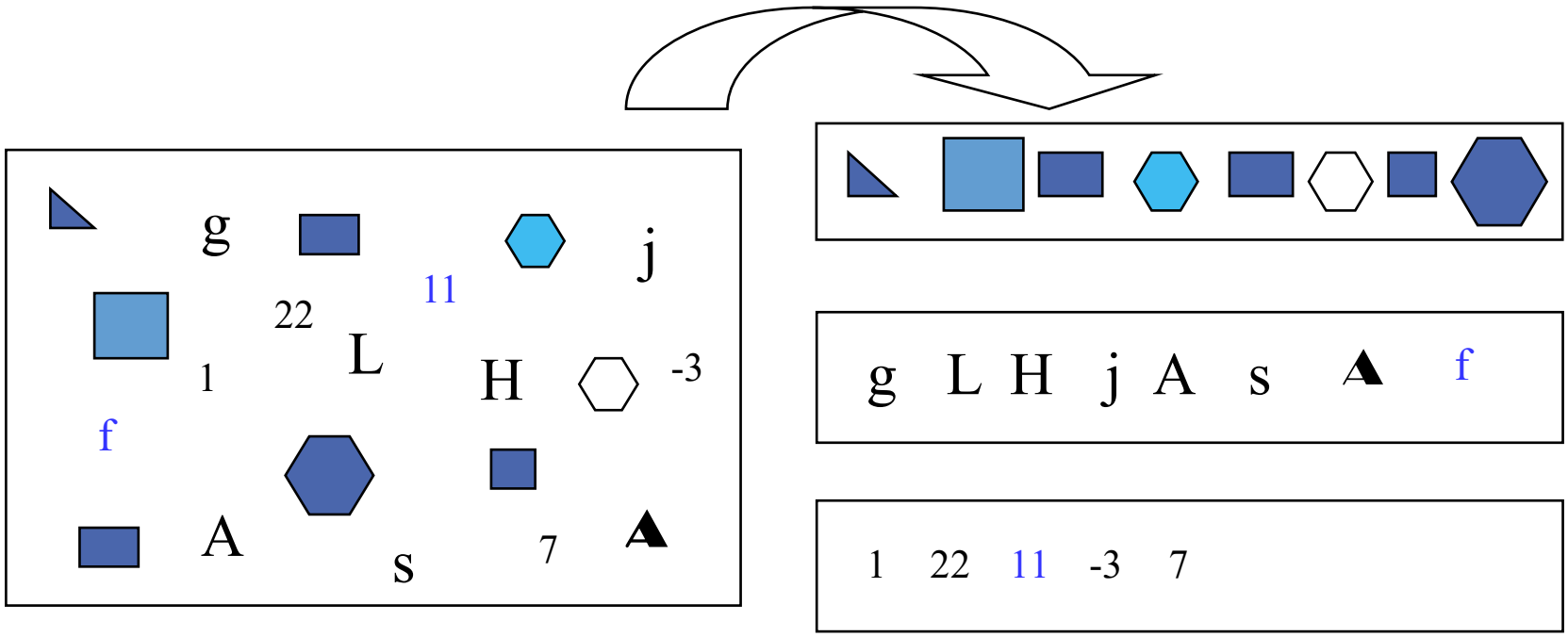


Der objektorientierte Ansatz

JAVACREAM
Training
Consulting
Projectmanagement

- Vorgehensweise
  - Klassifizieren
  - Abstrahieren
  - Ordnen, Bilden von Hierarchien
- Ein "menschlicher" Lösungsansatz!

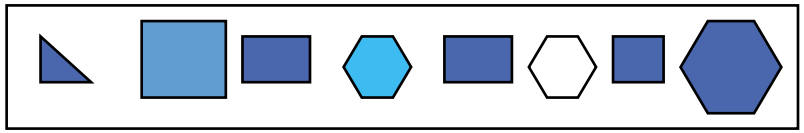
Klassifizieren



Abstrahieren

- Ein Zeichnungsobjekt hat eine Farbe, eine Position und eine Größe als Eigenschaften. Das komplexe Zeichnungsobjekt ist eine Komposition einfacherer Elemente.

Abstrahieren



Zeichnungsobjekte
- Farbe
- Position
- Größe

g L H j A s A f

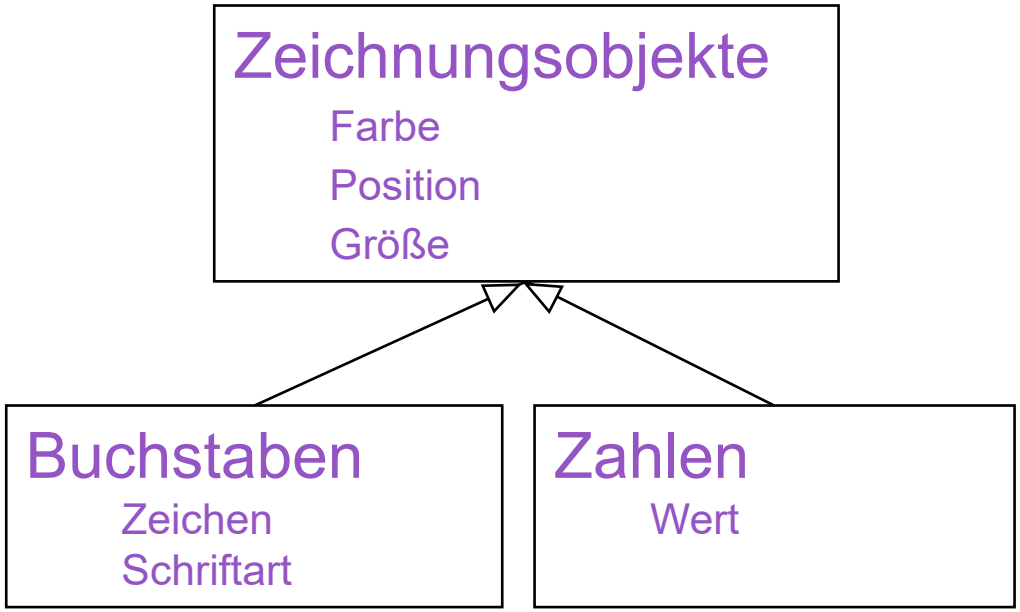
Buchstaben
- Zeichen
- Schriftart
- Farbe
- Position
- Größe

1 22 11 -3 7

Zahlen
- Wert
- Farbe
- Position
- Größe

Ordnen, Bilden von Hierarchien

- Ein Buchstabe ist ein Zeichnungsobjekt, das ein Zeichen in einer Schriftart darstellt
- Eine Zahl ist ein Zeichnungsobjekt, das einen Zahlenwert darstellt
- Buchstaben und Zahlen sind Zeichnungsobjekte und erben automatisch auch alle Eigenschaften eines Zeichnungsobjekts



Objekte

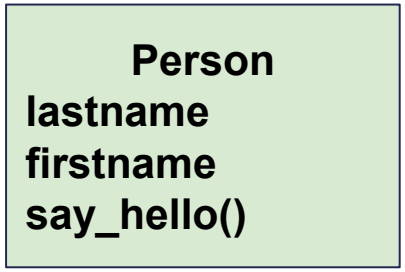
- Ein Objekt besitzt
  - Eigenschaften ⇒ Attribute
  - Fähigkeiten ⇒ Methoden
  - Interaktivität ⇒ Botschaften

- Analogie zu traditionellen Programmen
  - Attribute ⇔ Variable
  - Methoden ⇔ Funktionen, Prozeduren
  - Botschaften ⇔ Ablaufsteuerung, Parameter
- Gleichartige Objekte werden zu Klassen abstrahiert
  - Eine Klasse dient als Vorlage, Bauanleitung für (mehrere) Objekte

13.2

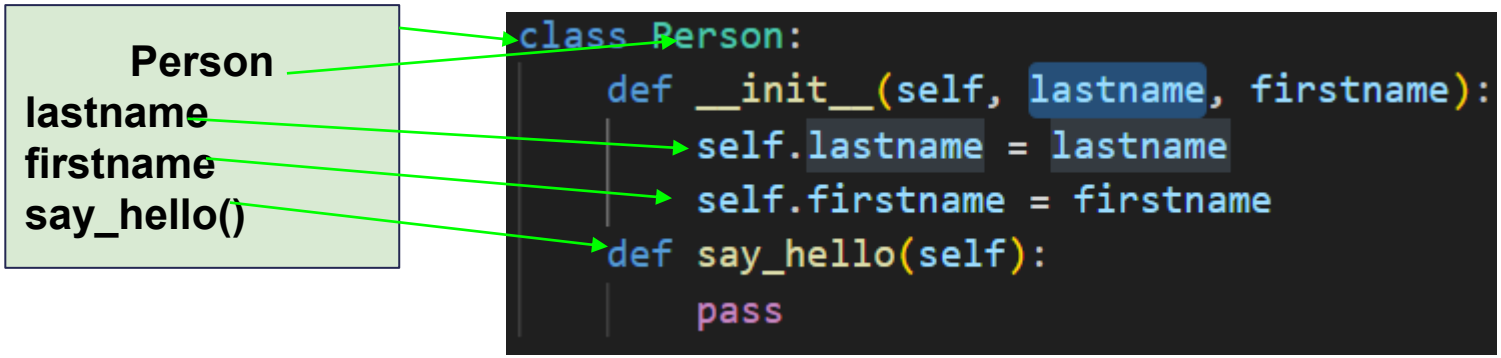
KLASSENMODELL UND UMSETZUNG IN PYTHON

Ein Modell visualisiert über ein Klassendiagramm



- In OOP werden Klassen programmiert
- Eine Klasse ist ein Template = Abstraktion von Objekten
- Ein Objekt ist eine Instanz einer Klasse

Umsetzung in Python



Erzeugung eines Personen-Objekts

- "Ich instanziiere die Klasse Person"
  - Objekt = Instanz einer Klasse

```
def main():
    person1 = Person("Sawitzki", "Rainer")
    print(type(person1))
```

- **Der Konstruktor __init__ hat drei Parameter, der Aufruf der Klasse als Funktion aber einen weniger**
  - **self ist das implizit erzeugte "leere" Objekt, dass durch den Konstruktor mit Eigenschaften befüllt wird**

Implementierung der Methoden

- Das ist nicht aus dem Klassendiagramm ersichtlich, hier wird eine zusätzliche fachliche Spezifikation notwendig sein
  - "Beim Aufruf von say_hello soll als Ergebnis eine Zeichenkette der Form"
    - 'Hallo, mein Name ist {firstname} {lastname}'

```
def say_hello(self):
    greeting = f'Hallo, mein Name ist {self.firstname} {self.lastname}'
    return greeting
```

- Aufruf

```
print(person1.say_hello())
```

Erweiterung

Fachliche Erweiterung: Eine Person hat Adressinformationen, city und street

Person
lastname
firstname
city
street
say_hello()

- **Diese Umsetzung ist nicht "gut":**
  - **Es können ja auch andere Dinge eine Adressinformation besitzen, diese müssten dann auch alle neue Attribute bekommen**
  - **Änderung in den Address-Eigenschaften (z.B. country) müssten dann mehrfach gemacht werden**

Erweiterung

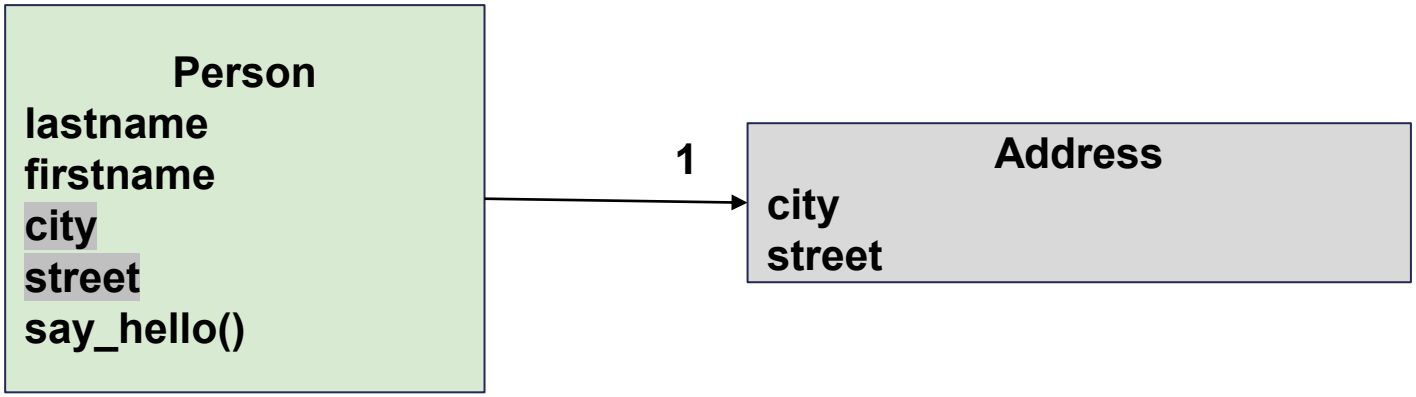
Fachliche Erweiterung: Eine Person hat Adressinformationen, city und street

Person
lastname
firstname
city
street
say_hello()

Address
city
street

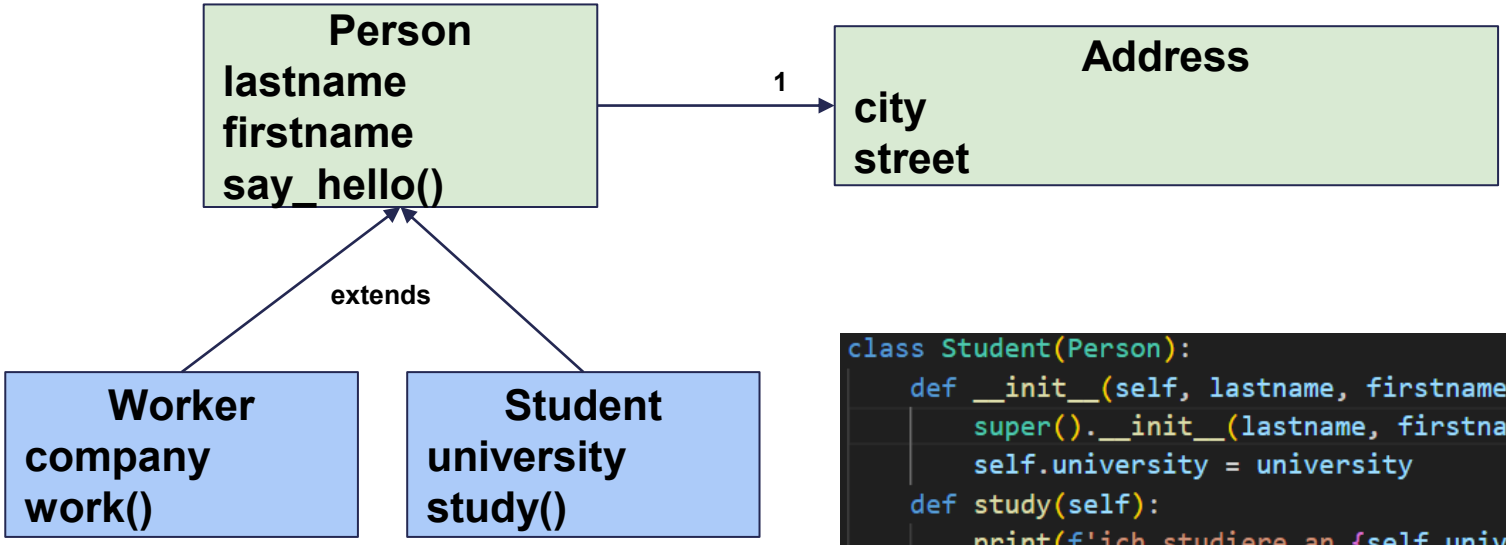
- **Viel besser, Addressinformationen bekommen eine eigene Klasse**
  - **Aber: Der Zusammenhang Person – Addresse ist dem Modell nicht zu entnehmen**

Fachliche Erweiterung: Eine Person hat Adressinformationen, city und street



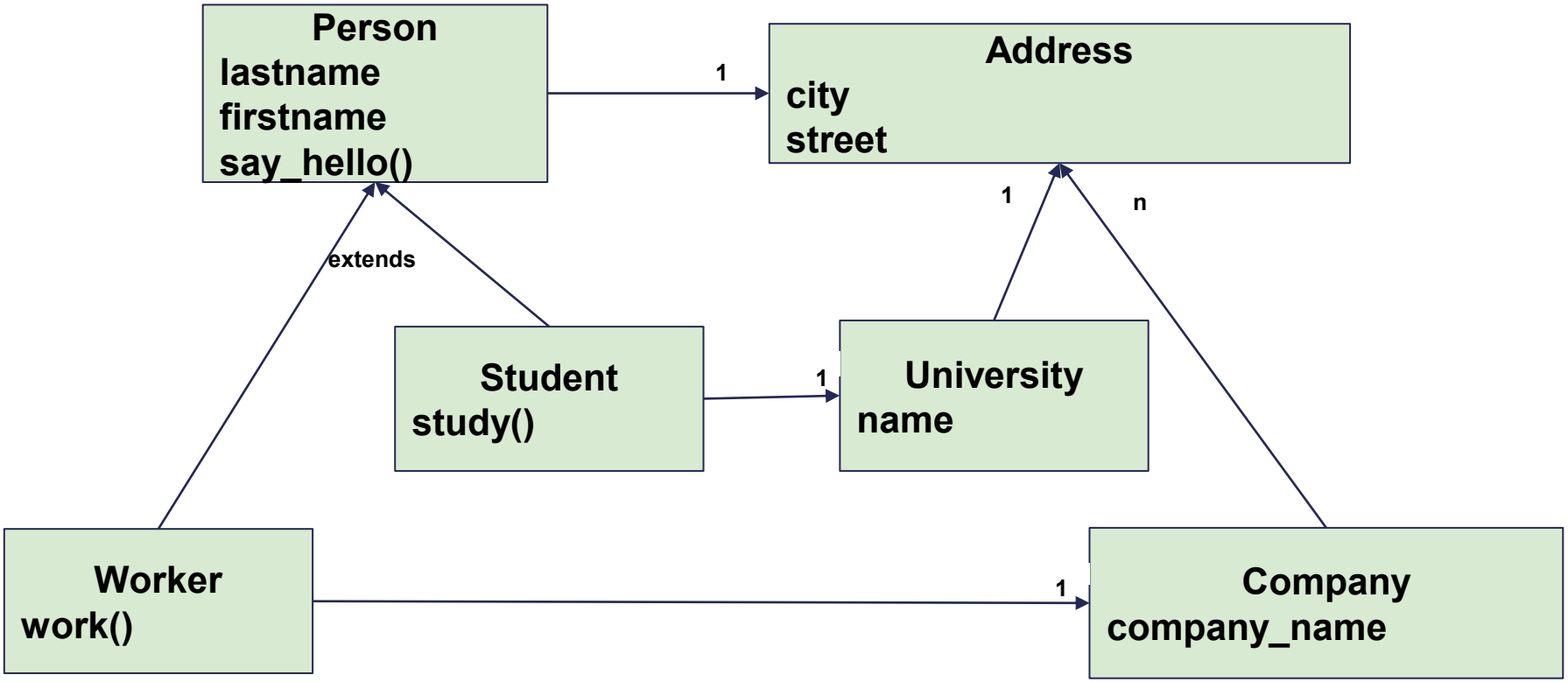
- Der Pfeil mit Zahl definiert die Abhängigkeit, die Zahl entspricht der Kardinalität
  - Hier: Eine Person hat extakt eine Addresse
  - Andere Möglichkeiten
    - 0:1 (eine oder keine)
    - 0:n (beliebig viele)
  - Die Richtung des Pfeils definiert die Richtung der Abhängigkeit
    - Hier: Person hat Addresse, die Addresse weiß nichts von Personen
    - Alternativ: Bidirektional mit Kardinalität

- Es gibt "Abarten" von Personen
  - Ein Student studiert an einer Universität
  - Ein Arbeiter arbeitet in einer Firma




```python
class Student(Person):
    def __init__(self, lastname, firstname, address, university):
        super().__init__(lastname, firstname, address)
        self.university = university
    def study(self):
        print(f'ich studiere an {self.university}')
```

Komplexere Modelle sind natürlich möglich



13.3

POLYMORPHIE

Polymorphie

- "Vielgesichtigkeit"
  - Eine Methode eines Objekts kann unterschiedliche Ausprägungen / Funktionalitäten enthalten
- Beispiel
  - Unsere Collections enthalten alle eine clear-Methode

Polymorphie in eigenen Klassen

- Beispiel
  - "Beim Aufruf von say_hello eines Studenten soll als Ergebnis eine Zeichenkette der Form"
    - 'Hallo, mein Name ist {firstname} {lastname} und ich studiere an {university}'
- Umsetzung
  - "Überschreiben" bzw. "Überschatten" von Methoden
    - Die Subklasse definiert nochmals eine Methode desselben Namens

12

DUNDER-METHODEN

12.1

MOTIVATION

- "Dunder" steht für "double underscore"-methods
  - manchmal auch als "magische Methoden" bezeichnet
- Dunders sind Methoden, die von BuiltIn-Funktionen, Operatoren oder "irgendwo" in Python-Bibliotheken benutzt werden
  - Beispiel
    - Die `print`-Funktion in Python ruft intern für das auszugebende Objekt die `__repr__`-Funktion auf
- Damit definieren Dunder-Methoden eine Art Vertrag zwischen einem allgemeinem Framework und einem Objekt
  - "Wenn Du mit print ausgegeben werden willst musst du mir die auszugebende Zeichenkette als Rückgabewert von `__repr__` geben"
  - "Wenn ich zwei Objekte auf Gleichheit prüfen möchte, übergebe ich das zu vergleichende Objekt deiner `__eq__`-Methode"

- **Welche Dunder-Methode wann und wie benutzt werden muss den jeweiligen Dokumentation entnommen werden**

__repr__ und __str__

- Diese beiden Methoden sollen beide Zeichenketten liefern
  - `__repr__`
    - Soll eine eindeutige und unmissverständliche Darstellung des Objekts liefern
    - Wird hauptsächlich für Debugging und Logging verwendet.
    - Wird direkt mit `repr(obj)` aufgerufen
  - `__str__`
    - Soll eine benutzerfreundliche und lesbare Darstellung liefern
    - Die Ausgabe sollte informativ und für den Endnutzer verständlich sein
    - Wird aufgerufen, wenn `str(obj)` oder `print(obj)` verwendet wird

- **Der Unterschied ist etwas akademisch, häufig genügt es, `__repr__` zu imlementieren**

__eq__ und __hash__

- `__eq__`
  - hat als Parameter das auf Gleichheit zu prüfende Objekt
  - Die Implementierung kann einfach
    - Beide Objekte mit `is` vergleichen und diesen Wert zurückgeben
      - Dann wird auf die Gleichheit der Referenzen geprüft
      - "Zwei Variablen sind gleich, wenn sie das identische Objekt referenzieren"+
    - Oder aber es wird auf Gleichheit eines oder mehrere Attribute geprüft
      - Dann wird die Gleichheit auf Inhalte bezogen
  - Werden Objekte mit == verglichen, ruft der Operator `__eq__` auf
- `__hash__`
  - Erzeugt intern einen Hash-Wert, der z.B. bei der Duplikatserkennung in einem Set oder bei Schlüsseln eines Dictionaries benutzt wird
  - Eine gute Hashfunktion berechnet den Hashwert möglichst disjunkt, wobei natürlich potenziell unterschiedliche Objekte den gleichen Hash produzieren können
    - Eine äußerst schlechte Hashfunktion wäre einfach ein `return 42`
      - Set und Dictionaries kommen damit schon zu recht, werden aber immer zunehmender Größe schnell ineffizient

Welche Strategie in einer eigenen Klasse gewählt wird muss der Aufgabenstellung entnommen werden

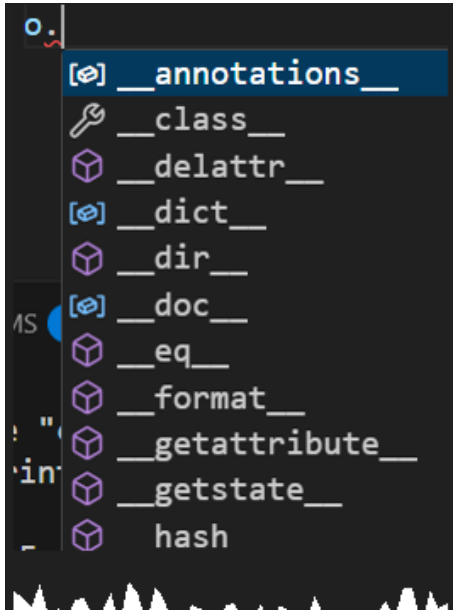
- **Eine Klasse, die __eq__ implementiert muss auch __hash__ implementieren**
  - **Sonst kann dieser Typ nicht beispielsweise in einem Set genutzt werden**
- **Zusätzlich muss sichergestellt sein, dass bei einem wahren __eq__-Vergleich beide Objekte auch denselben Hashwert produzieren**

12.2

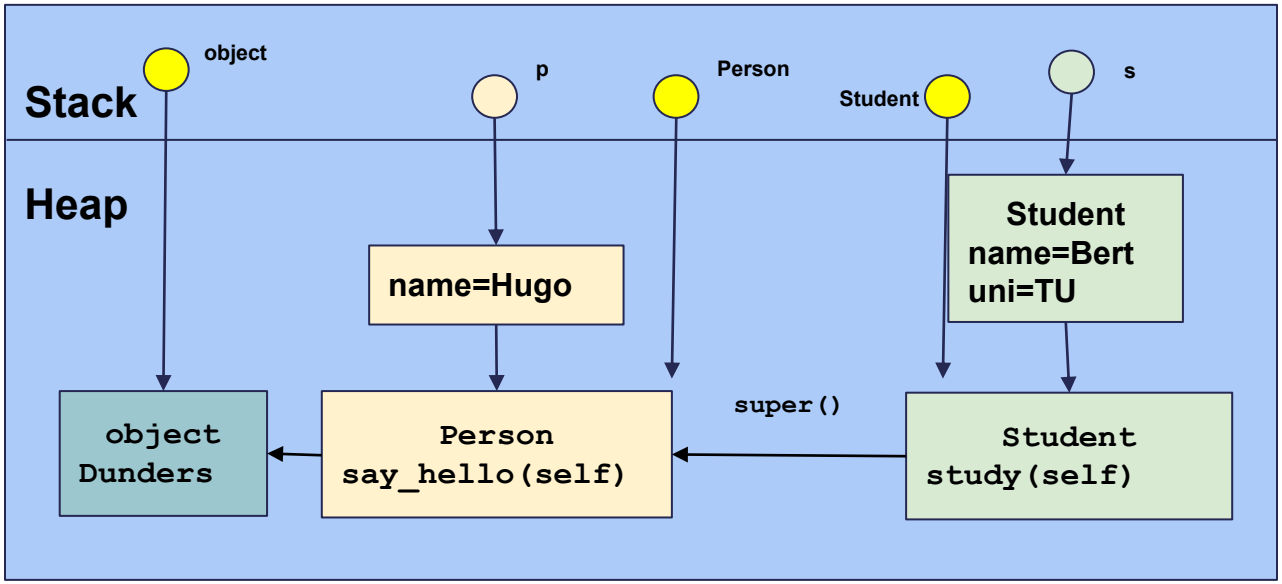
DIE OBJECT-KLASSE

Problem

- "Wir können doch Bücher, Autoren und Verleger mit `print` ausgeben, ohne `__repr__` zu implementieren. Warum gibt es keinen Fehler, wo kommt diese Methode denn her?"
- Die Lösung liegt an einem Detail der Vererbung
  - Jede selbstdefinierte Klasse hat, wenn nicht explizit eine Superklassen angegeben wird, die Klasse `object` als implizite Superklasse
  - Und diese Klasse definiert bereits einen Satz von Dunder-Methoden



object im Speicher



13

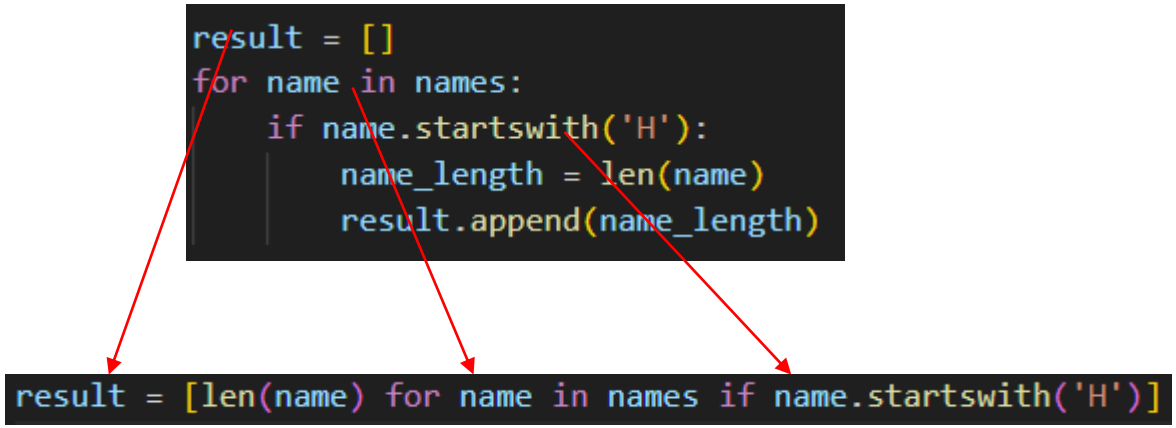
DATENVERARBEITUNG MIT COLLECTIONS

13.1

COMPREHENSIONS

Was sind Comprehension?

- Technische Antwort
  - "Eine Comprehension erzeugt aus einer potenziell verschachtelten Menge von iterierbaren Objekten eine Ergebnis-Collection"
- Pragmatische Antwort
  - "Eine Comprehension ist eine verkürzte und im Idealfall lesbarere Form einer Schleife, die anhand von Kriterien eine Ergebnis-Collection erstellt"
- Herleitung

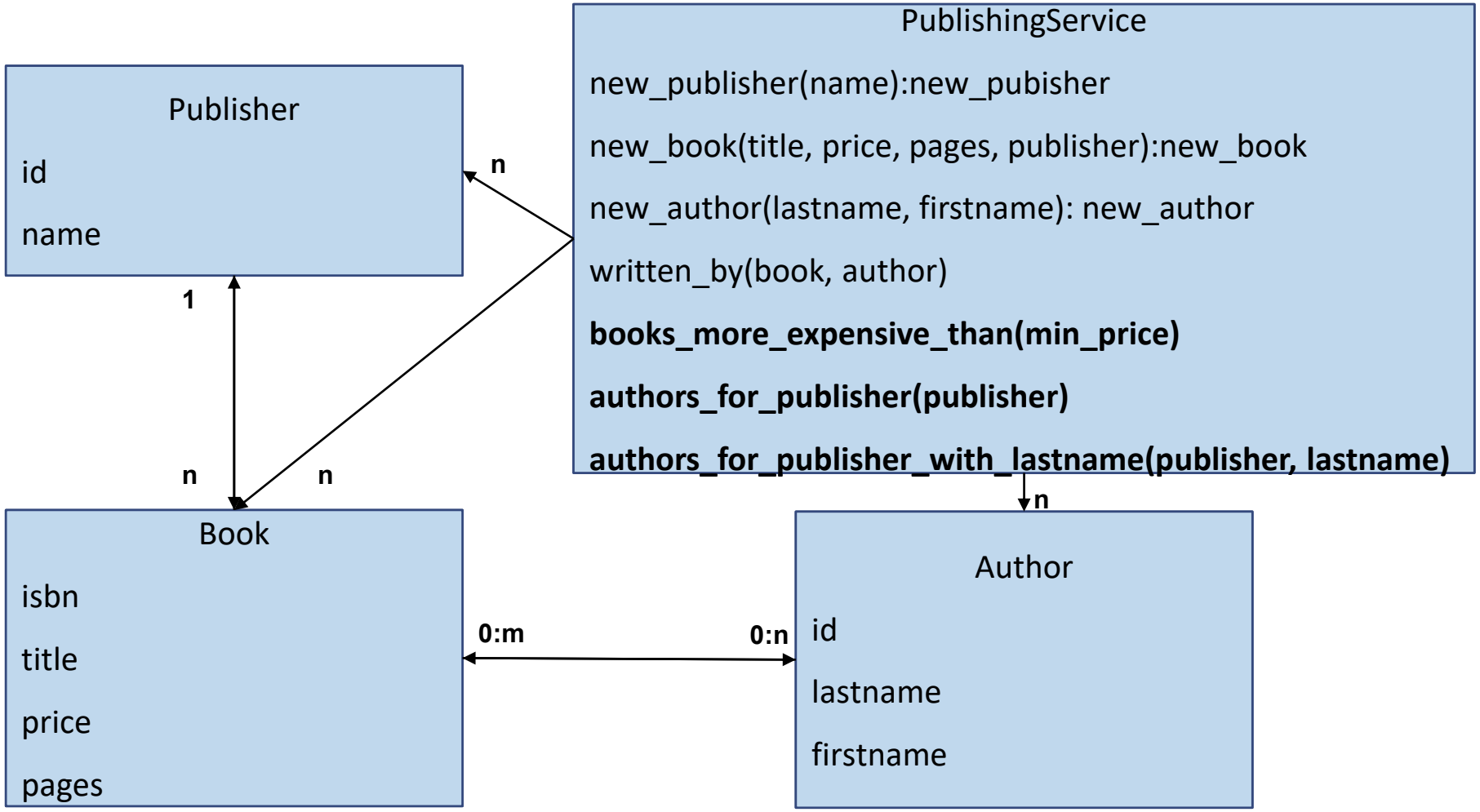


- **Ist die Lösung mit der Schleife lesbarer? -> Leave it up to you...**

Weitere Informationen

- Comprehensions sind in der Python-Community sehr beliebt und werden in vielen Anwendungen benutzt
  - Beispiele
    - https://www.w3schools.com/python/python_lists_comprehension.asp
- Es gibt Comprehensions für Ergebnis-Listen, -Sets und –Dictionaries
  - Also für alle Collection, die mit einem Literal erzeugt werden können
- Fragen wie "sind Comprehensions performanter oder effizienter?" sind müßig, beide Lösungen sind äquivalent
- Auch verschachtelte iterierbare Objekt-Aggregate können in einer Comprehension genutzt werden
  - Hierzu müssten verschachtelte Schleifen genutzt werden
- Comprehensions können nicht debugged werden

Publishing


PublishingService
new_publisher(name):new_pubisher
new_book(title, price, pages, publisher):new_book
new_author(lastname, firstname): new_author
written_by(book, author)
books_more_expensive_than(min_price)
authors_for_publisher(publisher)
authors_for_publisher_with_lastname(publisher, lastname)
Publisher
id
name
Book
isbn
title
price
pages
Author
id
lastname
firstname
n
1
n
n
n
0:m
0:n

13.2

LAMBDAS

Funktionen als Objekte

- Funktionen können Variablen zugewiesen werden

```
def f: .....
a=f
a("Hallo!")
```

- Funktionen können somit auch benutzt werden
  - als Parameter
  - als Rückgabewerte

Lambda-Funktionsdefinition

- Warum eine weitere Syntax für Funktionen
  - Das Schlüsselwort `def` kann nicht an beliebigen Stellen stehen
  - Eine Funktionsdefinition ist verbose (ausführlich)
    - Name, Parameterliste in runden Klammern, return-Statement
- `lambda` ist ein "Funktions-Literal"
- Lambda-Ausdrücke weisen einer Variable = Referent eine Funktion = Objekt zu

```
fkt=lambda x:x**2
fkt(2)
4
```

Weitere Informationen

- Einschränkungen
  - Lambdas sind in Python keine vollständigen Funktionen!
    - Sie dürfen nur eine einzige Anweisung enthalten
  - Lambda-Ausdrücke können nicht debugged werden
- Das Ergebnis der Anweisung ist der implizite Rückgabewert
- Lambda-Ausdrücke können auch innerhalb von Anweisungen stehen
  - Sehr praktisch für Parameter

```
my_func(lambda x:x**x)
```

- **Zur Klarstellung**
  - **Jede Funktion ist ein Objekt, das über eine Referenz angesprochen wird**
    - **Ob diese Funktion mit lambda oder def erzeugt wurde ist irrelevant**

16.2

HTTP-ZUGRIFF

- Ein Netzwerkzugriff-Zugriff erfordert eine große Menge von technisch aufwändigen Aktionen
  - Verbindungsaufbau zum Server durch Angabe einer IP-Adresse und einer Portnummer
  - Erstellen eines privaten Socket-Socket-Verbindung
  - Austausch von Daten in einer Form, die sowohl der Client als auch der Server verstehen
  - Dekodieren und Enkodieren der Daten
- Http vereinfacht diese Kommunikation
  - Portnummer 80
  - Verwendung eines standardisierten Containers mit Headern und Body
  - Durch diese Standardisierung können Module den Aufwand der Programmierung drastisch verringern
    - Im Endeffekt ein Einzeiler (!)

Schritt für Schritt

- Eine Recherche führt schnell zum requests-Modul
  - https://pypi.org/project/requests/
- Installation mit
  - pip install requests
- Aufruf einer Web-Seite mit z.B. Buchinformationen
  - https://openlibrary.org/dev/docs
- Ausführen einer Abfrage

17

DATENBANKZUGRIFF

17.1

ÜBERBLICK

- Datenbanken sind Produkte von Gerstellern und damit proprietär
  - Jeder Hersteller stellt ein Treibermodul zur Verfügung, um mit seiner Datenbank kommunizieren zu können
- Allerdings existiert für die gebräuchlichen Datenbanken (alle klassischen!) eine gemeinsame Programmiersprache
  - SQL
    - Structured Query Language
    - Ausgesprochen "sequel"

- **Hinweise**
  - **Die Ausführungen beziehen sich primär auf sogenannte "relationale Datenbanken", die in der Praxis sehr häufig benutzt werden**
  - **Es gibt jedoch auch andere Datenbanksysteme ("NoSql")**
    - **Diese werden zwar nicht mit SQL abgefragt, aber das Prinzip ist dasselbe**

17.2

SQL PRIMER

- Pragmatische Einführung in SQL mit Beschränkung auf Leseoperationen
  - w3school
  - https://www.w3schools.com/sql/

SQL Select
SQL Select Distinct
SQL Where
SQL Order By
SQL And
SQL Or
SQL Not

17.3

BEISPIEL

- Datenbank ist eine MySql-Datenbank
  - Notwendig ist damit die Installation des Treibers von MySql
    - pip install mysql-connector-python
  - Weitere Informationen
    - Datenbankname, User und Password müssen im Programm angepasst werden

18

RESTFUL WEBSERVICES

18.1

REKAPITULATION UND ÜBERBLICK

Rekapitulation

- Kommunikation auf Basis des Internet-Protokolls http / https
  - http basiert auf dem Standard TCP/IP-Stack
- Modul zur Nutzung des http-Protokolls
  - z.B. requests
- Client-Applikation ist (fast) ein Einzeiler
- RESTful WebServices liefern in den allermeisten Fällen die Daten im JSON-Format
- Umwandlung des JSON-Datenformats in die Python-Welt erfolgt automatisch
  - Daten liegen als Liste von Dictionaries vor

- Es gibt einen spezifizierten Satz von Header-Parameters
  - Content-Type (Request und Response)
    - Beschreibt, welches Datenformat dem Bytearray des Body zugrunde liegt
    - text/plain, text/html, application/pdf, application/json
  - Status (Response)
    - Zahl im Bereich 100-900
      - 200-299 -> Alles in Ordnung, "OK" = 200
      - 400-499 -> Fehlersituationen, die Daten betreffen, "Not Found" = 404
      - 500-599 -> Technische Fehler, "Informieren Sie den Server Administrator"
  - Method (Request)
    - Hinweis, was der Server mit den gesendeten Informationen machen soll
      - POST (Hinweis: Neue Ressource anlegen)
      - GET (Hinweis: Ressource lesen)
      - PUT / PATCH (Hinweis: Ressource aktualisieren)
      - DELETE (Hinweis: Ressource löschen)

18.2

BEISPIEL

Lesender Zugriff auf einen RESTful WebService

JAVACREAM
Training
Consulting
Projectmanagement

- Hier wird wieder das requests-Modul genutzt
- requests bietet für alle Http-Operationen Methoden
- Die Konvertierung von JSON erfolgt automatisch
  - JSON-Objekte werden Dictionaries
  - JSON-Listen Python-Listen

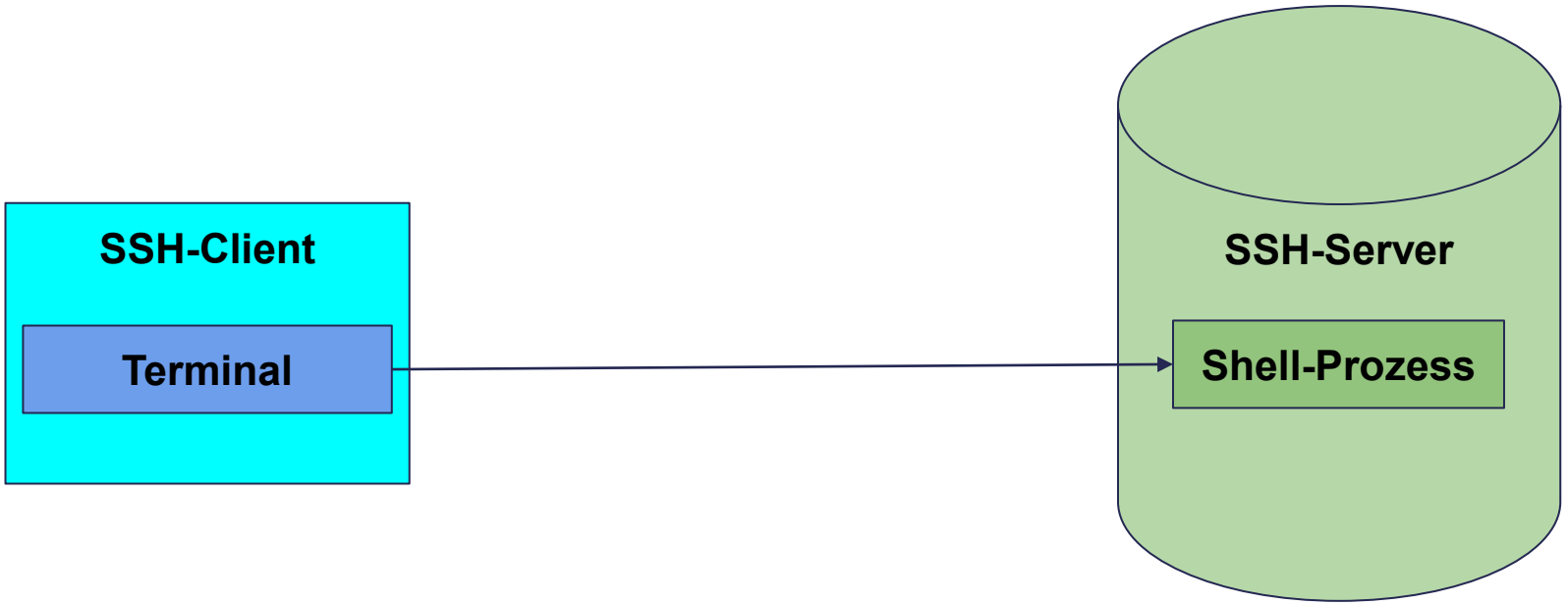
19

SSH

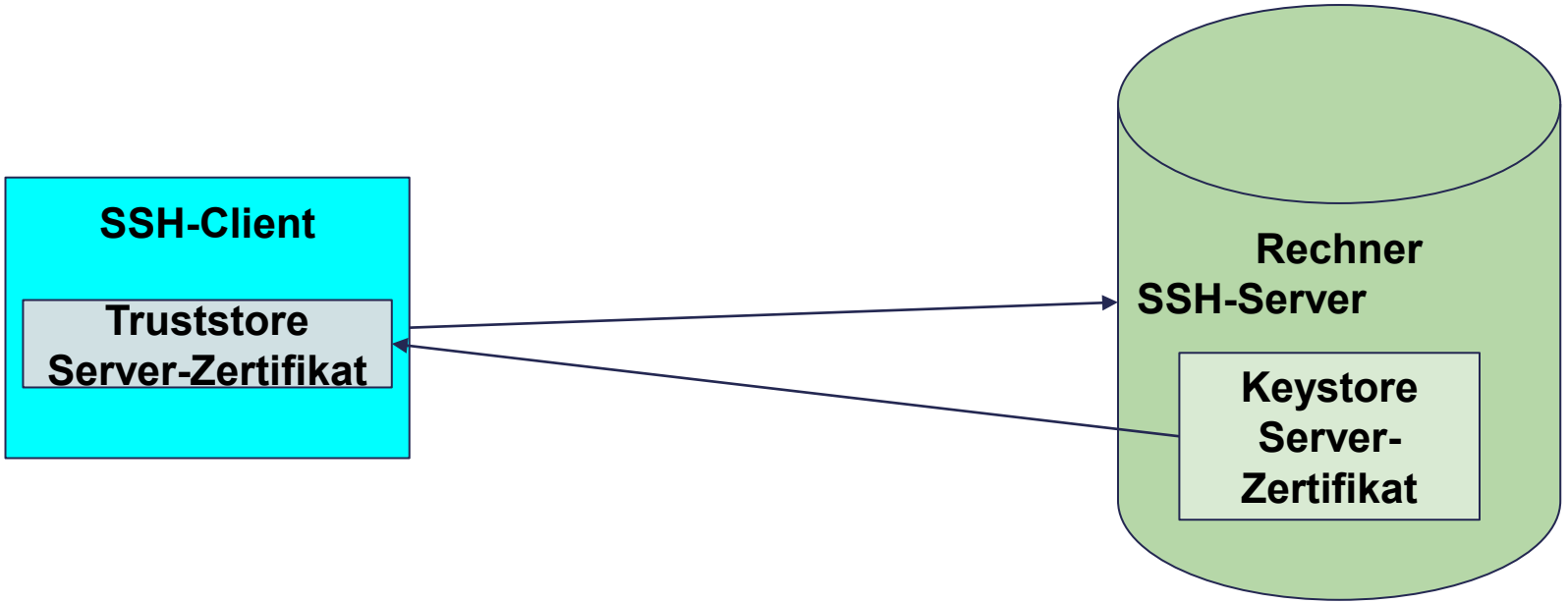
19.1

ÜBERSICHT

Prinzip

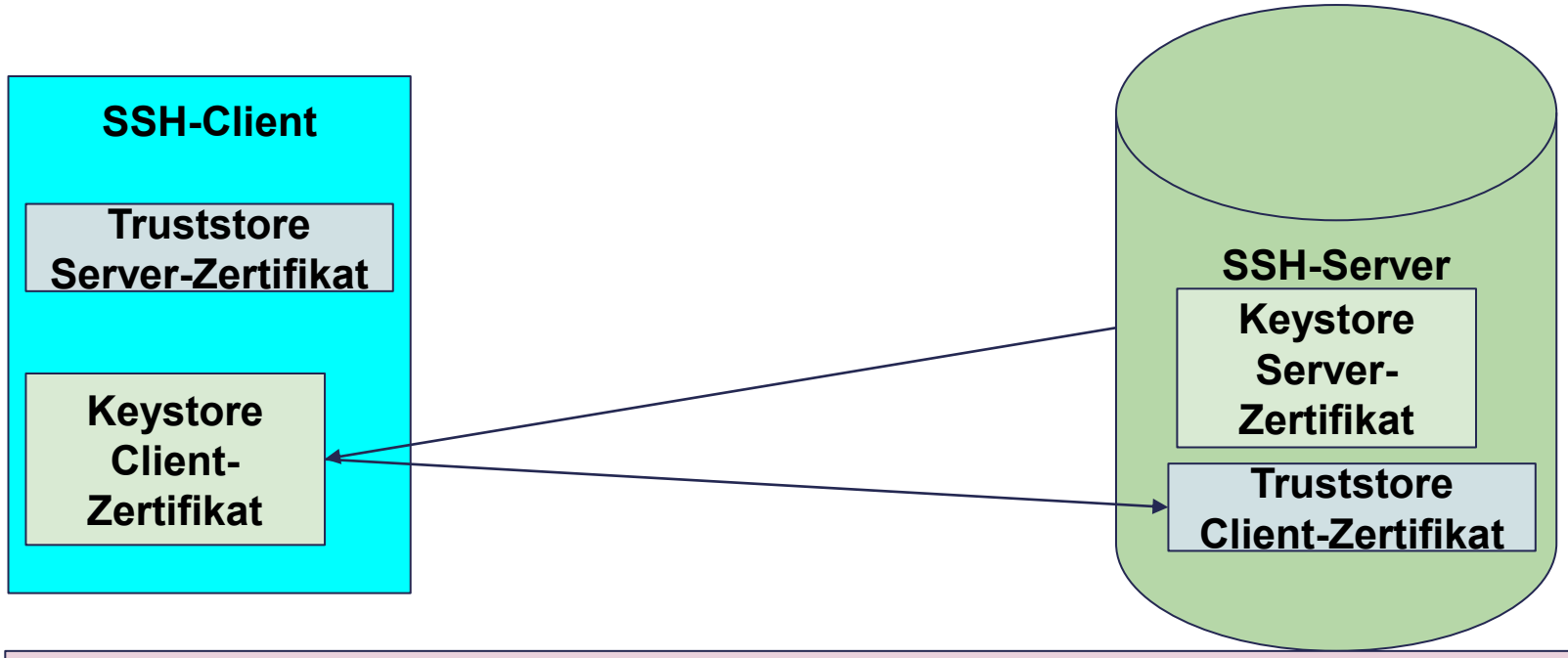


Was ist alles im ersten S von SSH drin?



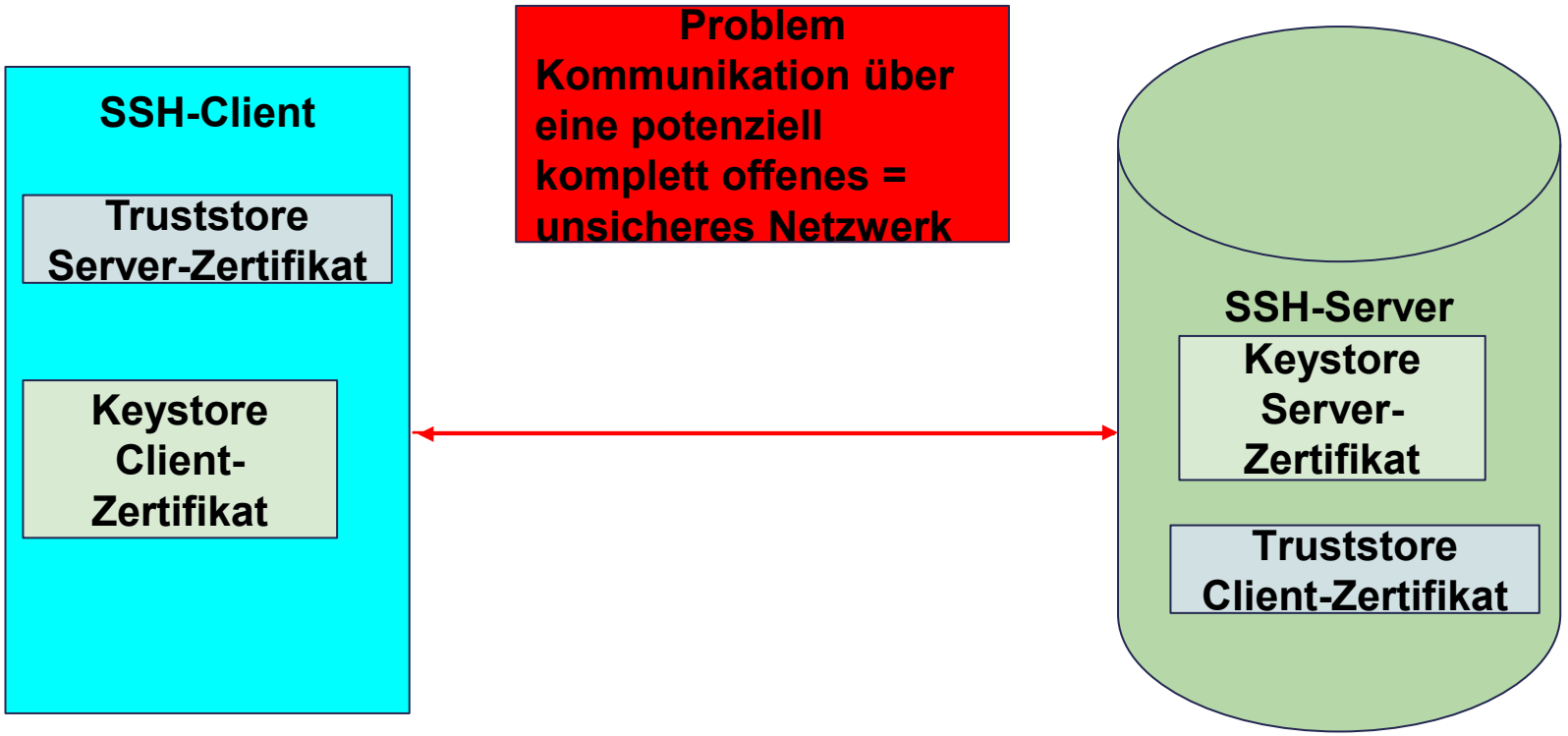
Schritt 1: Verifikation der Server-Identität
Bei Erfolg vertraut der Client dem SSH-Zertifikat des Servers und erlaubt Verbindungen zum Server

Was ist alles im ersten S von SSH drin?

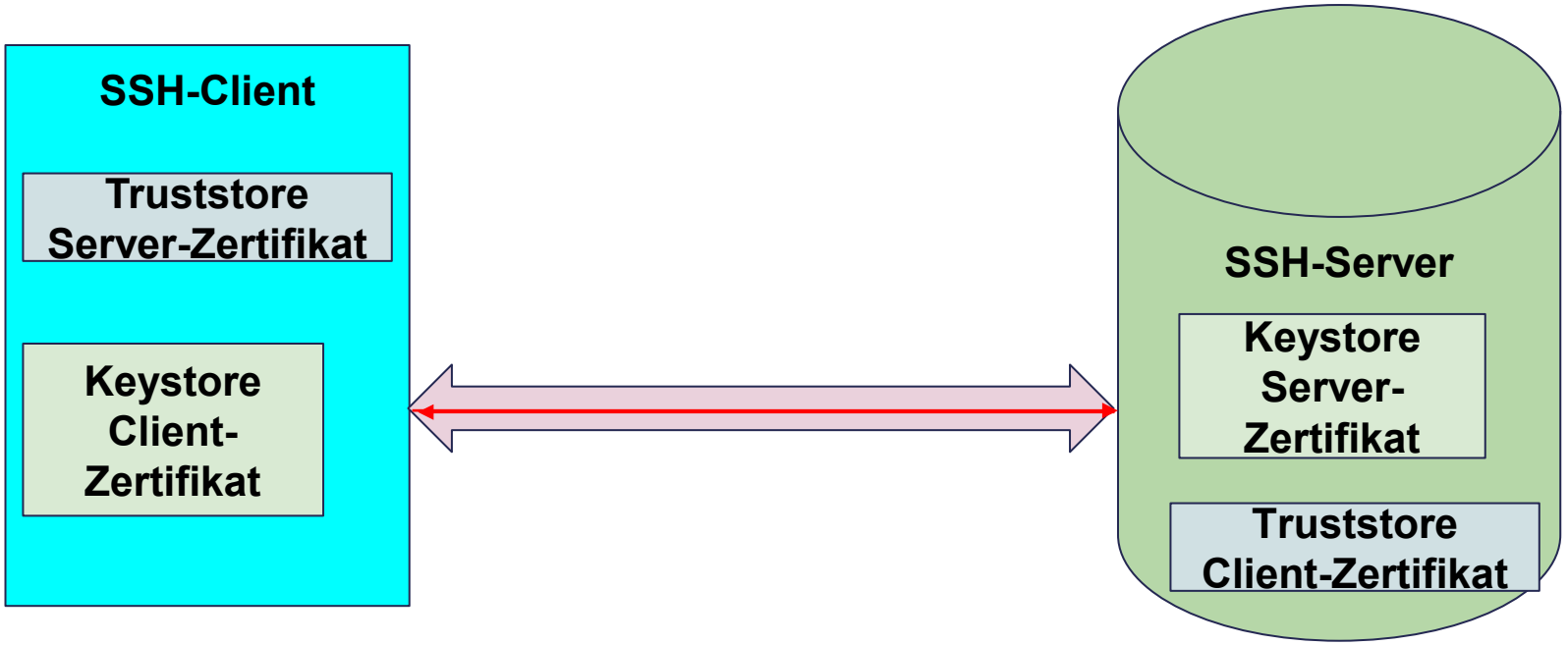


Schritt 1: Optional: Verifikation der Client-Identität
Bei Erfolg vertraut der Client dem SSH-Zertifikat des Servers und erlaubt Verbindungen zum Server

Was ist alles im ersten S von SSH drin?



Was ist alles im ersten S von SSH drin?



Schritt 2: Erzeugung und sicherer Austausch von Schlüsselpaaren zur Kommunikation

19.2

PARAMIKO

SSH-Module in Python

- Beispiel Paramiko
  - Eine sehr etablierte Bibliothek
- Nicht Bestandteil der Python-Standard-Umgebung
  - pip install paramiko
    - python -m pip install paramiko
- Sequenz
  - Aufbau der Verbindung
  - Nutzen der Verbindung
  - Schließen der Verbindung

Ein paar Hinweise

- Der Paramiko-Client nutzt für jedes execute einen neuen Shell-Prozess
  - Beispielsweise Befehle mit cd verhalten sich "unerwartet"
- Password im Klartext des Scripts ist natürlich Unsinn
  - Sichere Ablage der Credentials z.B. mit keyring
- Fehlerbehandlung
  - SSH-Fehler wie
    - Falscher Host, Port
    - Falsche Credentials
  - stderr unbedingt ebenfalls auslesen und dekodieren