Deutsche Hochschule
für Prävention und Gesundheitsmanagement
University of Applied Sciences

Studienbrief

Programmierung I - Objektorientierte Programmierung

Inhaltsverzeichnis

1 Grundlagen

Lernziele

Nach Bearbeitung des Kapitels *Grundlagen* ...

- wissen Sie, wie Informationen in einem Computer gespeichert werden.
- kennen Sie Begriffe Compiler, Interpreter und Debugger.
- kennen Sie die Grundsätze von funktionaler, reaktiver und objektorientierter Programmierung
- haben Sie einen Überblick über die klassischen Programmiersprachen.
- wissen Sie, was ein Algorithmus ist und sind in der Lage diese als Pseudo-Code geschriebene Programmabläufe zu lesen.

In diesem Teil lernen Sie die Grundlagen der Programmierung kennen. Dabei werden Sie lernen, wie Informationen in Computersystemen verarbeitet und dargestellt werden. Sie lernen darüber hinaus die gängigsten Paradigmen der Programmierung kennen und können Programmabläufe und Algorithmen strukturiert und anschaulich darstellen.

Computer sind sehr mächtige Werkzeuge. Da sie jedoch nicht selbständig denken können, müssen ihnen Instruktionen in Form von Computerprogrammen mitgeteilt werden. Ein interessanter Unterschied zwischen einem digitalen Taschenrechner und einem Computer ist dabei, dass man dem Computer im Gegensatz zum Taschenrechner neue Befehle beibringen kann. Ein *Programm* ist also eine Sammlung von Instruktionen. Unter dem Begriff ‚Programmierung' versteht man die Tätigkeit solche Programme zu erstellen.

Die Grundessenz der Programmierung neue Instruktionen und Operationen zu kreieren und diese zu nützlichen Funktionalitäten zu kombinieren. Die Computerprogramme werden dabei mit Hilfe einer Programmiersprache als Quellcode formuliert und in durch den Computer ausführbare Anweisungen (*Binärcode*) übersetzt. Dadurch übermittelt man dem Computer Instruktionen in einer Sprache, die er versteht. Im weitesten Sinne umfasst die Programmierung dabei auch das Testen sowie die Dokumentation der Programme.

1.6 Was brauche ich für die Python-Programmierung?

Für die Bearbeitung der Beispiele und Übungen benötigen Sie lediglich *Microsoft Visual Studio Code* mit der Python-Erweiterung sowie gegebenenfalls die Jupyter-Erweiterung. Mit Visual Studio (VS) Code und der Python-Erweiterung haben Sie alles was Sie brauchen für die Entwicklung von Python-Programmen im Rahmen dieses Studienmoduls.

1.6.1 Installieren & Konfigurieren von VS Code für Python

Die Installation von Visual Studio Code ist auf jeder Plattform recht einfach. Vollständige Anleitungen für Windows, Mac und Linux sind online verfügbar. Sie finden alles auf der Visual Studio Code-Website[29].

Falls Sie sich wundern, Visual Studio Code (oder kurz *VS Code*) hat mit seinem größeren Windows-basierten Namensvetter, Visual Studio, fast nur den Namen gemeinsam. Visual Studio Code bietet integrierte Unterstützung für mehrere Programmiersprachen und ein Erweiterungsmodell mit einem reichhaltigen Ökosystem an Unterstützung für andere Sprachen. VS Code wird monatlich aktualisiert, und Sie können sich im Microsoft Python Blog[30] auf dem Laufenden halten. Die Benutzeroberfläche von VS Code ist gut dokumentiert, so dass sie hier nicht noch einmal wiederholt wird.

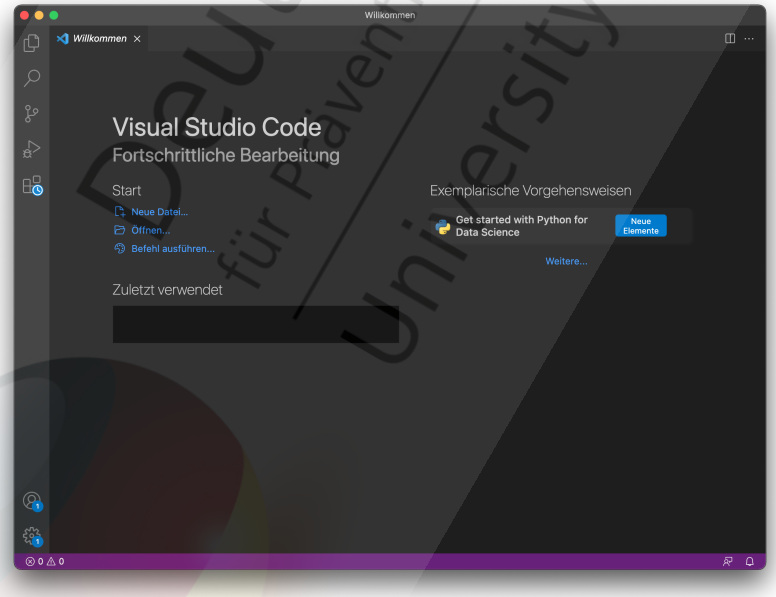


Abbildung 1: Beispielabbildung des Startfenster von VS Code. (©BSA/DHfPG)

[29] https://code.visualstudio.com/
[30] http://aka.ms/pythonblog

2.2.1 Numerische Datentypen

In Python unterscheidet man zwischen den folgenden numerischen Datentypen:

- Ganze Zahlen: `int`
- Gleitkommazahlen: `float`
- Boolsche Werte: `bool`

Neben diesen drei genannten existieren noch weitere, wie zum Beispiel für die komplexen Zahlen. Für dieses Modul und grundsätzlich auch den Studiengang reichen die numerischen Datentypen `int`, `long`, `float` und `bool` aus. Der Datentyp `int` beinhaltet die ganzen Zahlen, wie zum Beispiel 2, 3, 10, 100, 10000000, −5 oder −10000. `float` steht für die Dezimalzahlen (Kommazahlen, Gleit- oder auch Fließkommazahlen), wie zum Beispiel 3.14, 1.9999, -0.96424 oder 10.23. Beachten Sie unbedingt, dass in allen gängigen Programmiersprachen der Punkt statt des Kommas als Dezimaltrennzeichen verwendet wird (englische Konvention). Die zulässige Größe ist im Gegensatz zu vielen anderen Programmiersprachen in Python nicht direkt beschränkt, sondern hängt nur vom Speicherplatz ab.

Wie im vorherigen Abschnitt geschrieben, muss man sich in Python nicht explizit um den Datentyp kümmern, das heißt wenn man `a = 3` eingibt, dann wird `a` automatisch eine Variable vom Typ `int`. Wenn Sie `a = 3.0` schreiben, wird `a` eine Variable vom Typ `float`. Gleitkommazahlen vom Typ `float` können auch in exponentieller Form eingegeben werden, wie zum Beispiel `2e5` für 200000 oder `1e-3` für 0.001. Bei `a = True` bekommt `a` folglich den Typ `bool`. Umwandlungen eines zu konvertierenden Werts `a` in einen neuen Typ werden allgemein in der Form `neuerTyp(a)` geschrieben, wie zum Beispiel `int(a)` oder `bool(a)`.

Codebeispiel -

```
type(1)       # Output: <type 'int'>
type(1.0)     # Output: <type 'float'>
type(True)    # Output: <type 'bool'>

a = int(1.0)
type(a)       # Output: <type 'int'>
a = bool(0)
type(a)       # Output: <type 'bool'>

bool(1)       # Output: True

```

Codebeispiel - Anführungszeichen in Dictionaries

Wenn Sie um die Schlüssel des Dictionaries die gleiche Art von Anführungszeichen verwenden wie an der Außenseite des f-Strings, wird das Anführungszeichen am Anfang des ersten Schlüssels des Dictionaries als Ende der Zeichenkette interpretiert.

```
>>> kunde = {'name':'Max Mustermann', 'alter': 34}
>>> print(f'Der Kunde heißt {kunde['name']} und ist {
 kunde['alter']} Jahre alt.')
File "<stdin>", line 1
s = f'Der Kunde heißt {kunde['name']} und ist {kunde['
 alter']} Jahre alt.'
^
SyntaxError: invalid syntax

```

Normalerweise wählt man für die f-Strings immer doppelte Anführungszeichen und für die Schlüssel des Dictionaries einfache Anführungszeichen. Korrekterweise müssten also die Anführungszeichen des f-Strings und der Schlüssel des Dictionaries unterschiedlich sein:

```
>>> kunde = {'name':'Max Mustermann', 'alter': 34}
>>> print(f"Der Kunde heißt {kunde['name']} und ist {
 kunde['alter']} Jahre alt.")
Der Kunde heißt Max Mustermann und ist 34 Jahre alt.

```

Sie haben bereits den Backslash als Escape-Zeichen für Strings kennen gelernt. Leider können Sie aber diese nicht in f-Strings verwenden. Ein Workaround wäre hier, den String mit den nicht zulässigen Zeichen vorher in einer Variable mit Escape-Zeichen zu deklarieren:

```
>>> name = "\#Max \"Mustermann\""
>>> f"{name}"
'#Max "Mustermann"'
```

Um nun auch geschweifte Klammern oder Anführungszeichen in f-Strings darzustellen, verhält es sich ähnlich. Um geschweifte Klammern in f-Strings darzustellen, müssen Sie lediglich die Klammern doppeln:

```
>>> f"{{40 + 2}}"
'{40 + 2}'
```

Dreifache Klammern lassen den Ausdruck wieder zu und stellen einfache Klammern zusätzlich dar:

3.4 Dictionaries

Die nächste sequentielle Datenstruktur, die Sie in diesem Studienbrief kennen lernen, ist das sogenannte *Dictionary*. Man kann den Begriff mit „Wörterbuch" ins Deutsche übersetzen, doch würde dies eher für Verwirrung sorgen, da man unter Programmierenden doch eher die englischen Begriffe verwendet. Dictionaries sind spezielle digitale Sammlungen, um Datenwerte in Schlüssel:Wert-Paare (oder *key:value*-Paare) zu speichern.

Ein Dictionary ist in Python seit Version 3.7 geordnet, veränderbar und erlaubt keine Duplikate, das heißt, es ist jeder Schlüssel nur einmal innerhalb eines Dictionaries zulässig. Wird ein Schlüssel dennoch mehrfach verwendet, wird automatisch der letzte Wert im Dictionary genommen und alle vorherigen überschrieben. Geordnete Datenstrukturen lassen sich bekanntermaßen über einen Index ansprechen, besitzen also auch eine Reihenfolge, die sich nicht verändert (wie bei Mengen zum Beispiel). Dictionaries werden mit geschweiften Klammern umschlossen und besitzen darin kommagetrennte *key:value*-Paare, wovon der Schlüssel (links vom Doppelpunkt) immer in doppelte Anführungszeichen gesetzt werden muss. Die Werte rechts des Doppelpunktes können jeden beliebigen Datentyp annehmen, wie zum Beispiel Strings, ganze Zahlen, Listen oder sogar weitere verschachtelte Dictionaries.

```
person = {
  "vorname": "Max",
  "nachname": "Mustermann"
}

auto = {
  "marke": "Ford",
  "modell": "Mustang",
  "elektrisch": False,
  "farben": ["rot", "weiss", "blau"],
  "jahr": 1964,
  "besitzer": person
}
```

Dictionaries haben eine weitere spezielle Eigenschaft, die das Kopieren zweier Dictionaries betrifft. Sie können Dictionaries nicht einfach kopieren, indem Sie zum Beispiel `dict2 = dict1` schreiben. In diesem Fall wäre `dict2` nur ein Verweis (oder Referenz) auf `dict1`, und Änderungen, die Sie in `dict1` vornehmen, würden automatisch auch in `dict2` passieren. Es gibt jedoch verschiedene Möglichkeiten in Python eine Kopie eines Dictionaries zu erstellen. Eine Möglichkeit ist zum Beispiel die in Python integrierte `copy()`-Methode. Eine andere Möglichkeit wäre es die eingebaute `dict()`-Methode zu verwenden, die ein neues Objekt mithilfe des Dictionaries als Parameter erstellt.

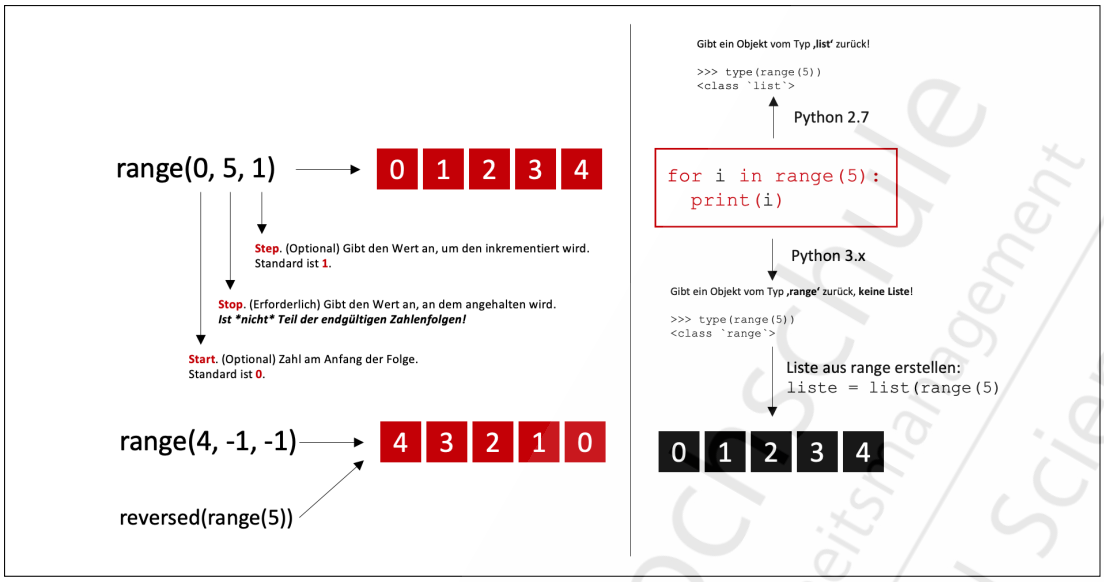


Abbildung 7: `range()` gibt die unveränderbare Zahlenfolge aus, beginnend beim gegebenen Start (`int`) bis zum Stopp-Schritt (`int`). Jede Zahl entspricht in jedem weiteren Schritt der Addition des Schrittwerts (`step`, *optional*) und der Zahl der vorherigen Schritts. (©BSA/DHfPG)

4.3.3 Die for-Schleife

Sie haben nun anhand der `while`-Schleife das Prinzip der Schleifen in der Programmierung kennen gelernt. Grundsätzlich geht es dabei um das Vereinfachen von Code. Im Grunde liefern folgende Codeblöcke das Gleiche Ergebnis:

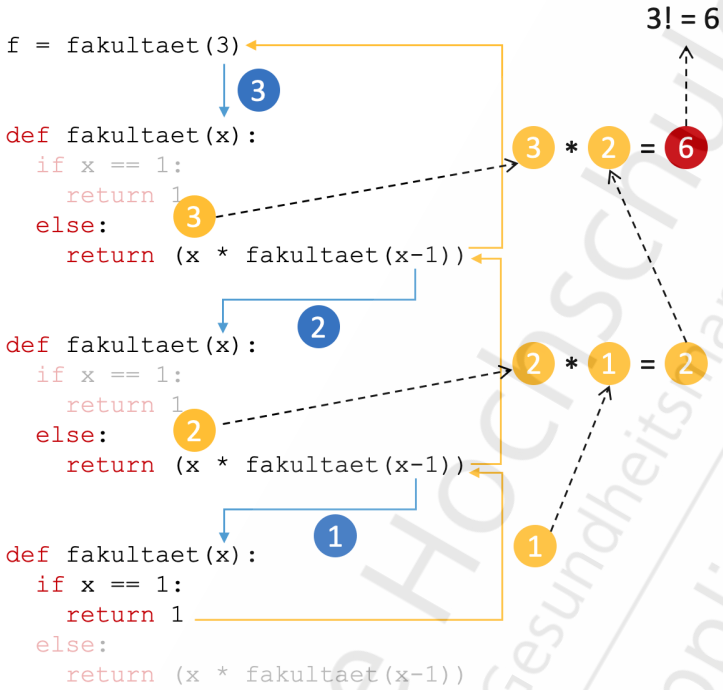
```
# Sequentiell hintereinander
a = 1
a += 1
a += 1
a += 1
a += 1
a += 1
a += 1
a += 1

print(a) # Output: 8

# Mithilfe einer while-Schleife
a = 1
while a < 8:
  a += 1

print(a) # Output: 8
```

Betrachten Sie zur weiteren Erklärung folgendes Schaubild:



Die Rekursion endet, wenn sich die Zahl auf 1 reduziert. Dies wird als *Basisbedingung* (oder Abbruchbedingung) bezeichnet. Jede rekursive Funktion muss eine solche Bedingung besitzen, die die Rekursion beendet. Sonst ruft sich die Funktion unendlich oft selbst auf. Der Python-Interpreter begrenzt die Tiefe der Rekursion auf 1000, um unendliche Rekursionen zu vermeiden, die zu Stack-Überläufen führen.

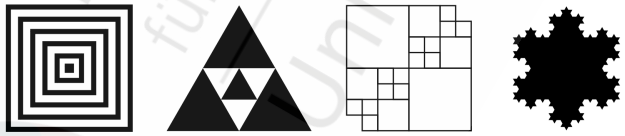


Abbildung 9: Beispielabbildungen von Rekursion. (©BSA/DHfPG)

In der Programmierung sollte man aber bei der Rekursion sehr vorsichtig sein, da es leicht passieren kann, dass man eine Funktion schreibt, die nie beendet wird, oder eine, die übermäßig viel Speicher oder Prozessorleistung verbraucht. Das Problem der Endlosschleife haben Sie schon bei der `while`-Schleife oder beim exponentiellen Wachstum in der Mathematik kennen gelernt. Wenn sie jedoch richtig geschrieben ist, kann die Rekursion ein sehr effizienter und mathematisch eleganter Ansatz zur Programmierung sein.

Alters von Chantale zurückgeben wird. Eine gute Möglichkeit, diese Art von Code überschaubarer und besser wartbar zu machen, ist die Verwendung von *Klassen*.

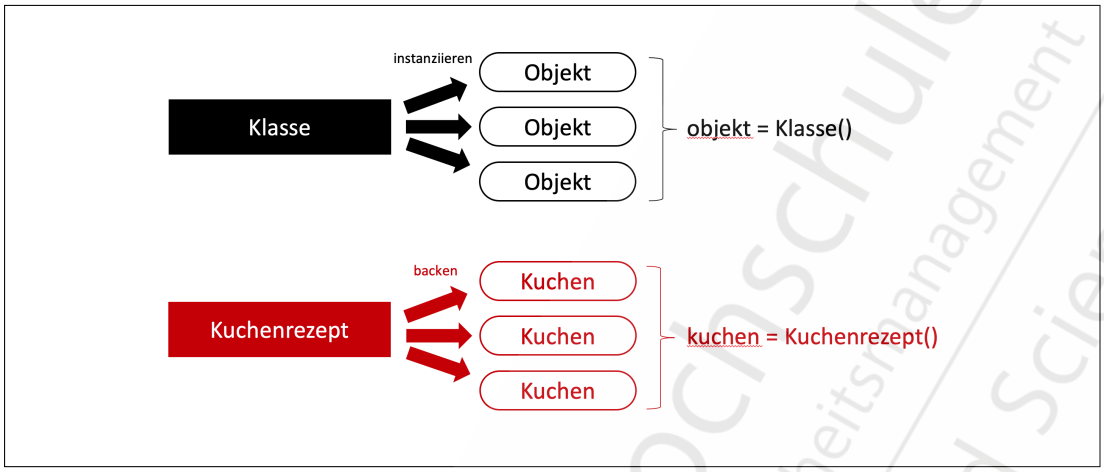


Abbildung 10: Analogie zwischen Klasse/Objekt und Kuchenrezept/Kuchen. (©BSA/DHfPG)

Klassen werden verwendet, um benutzerdefinierte Datenstrukturen zu erstellen. Klassen definieren Funktionen, die wie bei Objekten auch Methoden genannt werden und die Verhaltensweisen und Aktionen festlegen, die ein aus der Klasse erstelltes Objekt mit seinen Daten ausführen kann. Im Folgenden Beispiel werden Sie eine Klasse `Mitglied` erstellen, die einige Informationen über die Eigenschaften und Verhaltensweisen speichert, die eine einzelne Katze haben kann.

Eine Klasse ist eine Blaupause, Beschreibung oder Vorgabe dafür, wie etwas definiert werden soll, und enthält selbst eigentlich keine Daten. Die Klasse `Mitglied` legt fest, dass ein Name und ein Alter für die Definition eines Mitglieds eines Sportvereins erforderlich sind, aber sie enthält nicht den Namen oder das Alter eines bestimmten Mitglieds. Während die Klasse nur die Anleitung ist, ist eine *Instanz* ein Objekt, das aus einer Klasse aufgebaut ist und reale Daten und Werte enthält. Es ist ein tatsächliches Mitglied, zum Beispiel mit einem Namen Anna, welches 36 Jahre alt ist.

Anders ausgedrückt, eine Klasse ist wie eine Art Formular oder ein Fragebogen. Eine Instanz ist also wie ein Mitgliederformular, das mit Informationen über das neue Mitglied ausgefüllt wurde. Genauso wie viele Personen dasselbe Formular mit ihren eigenen, einzigartigen Informationen ausfüllen können, können viele Instanzen aus einer einzigen Klasse erstellt werden.

Alle Klassendefinitionen beginnen mit dem Schlüsselwort `class`, dem der Name der Klasse und ein Doppelpunkt folgen, so wie es in Python auch bei den Kontrollstrukturen wie Schleifen oder Funktionen üblich ist. Jeder Code, der unterhalb der Klassendefinition eingerückt ist, wird als Teil des Klassenkörpers betrachtet. Python-Klassennamen werden per Konvention in UpperCamelCase-Notation geschrieben. Zum Beispiel würde eine Klasse für eine bestimmte Hunderasse wie den Jack Russell Terrier als `JackRussellTerrier`

hergestellt werden. Das verhindert unnötiges Blockieren des Servers und verringert den Datenverkehr um unnötige Nachrichten.

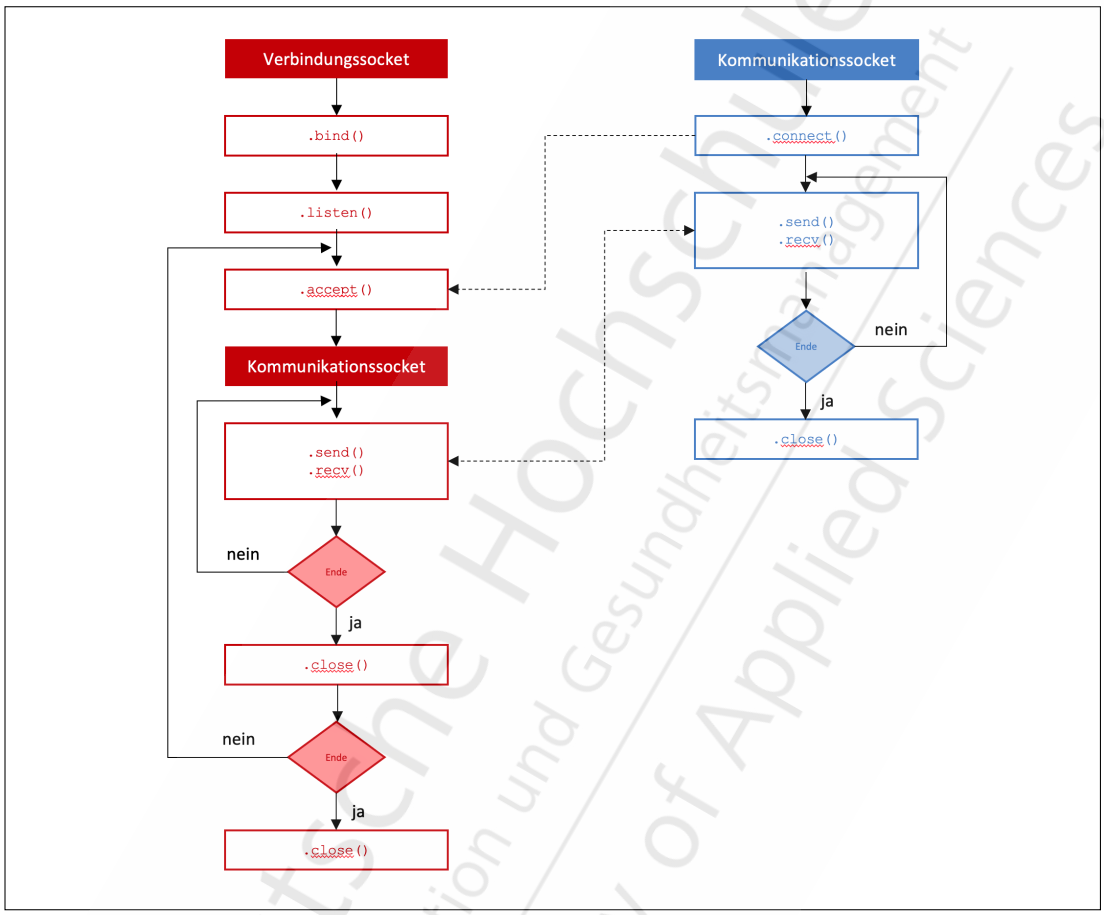


Abbildung 15: Das Client-Server-Modell. (©BSA/DHfPG)

Nachdem eine Verbindungsanfrage eingetroffen ist und mit `accept()` akzeptiert wurde, wird ein neuer Socket, der sogenannte *Kommunikationssocket*, erzeugt. Über einen solchen Kommunikationssocket wird die vollständige Kommunikation zwischen Server und Client über Methoden wie `send()` (zum Senden) oder `recv()` (zum Empfangen) abgewickelt. Ein Kommunikationssocket ist immer nur für einen verbundenen Client zuständig.

Sobald die Kommunikation beendet ist, wird das Kommunikationsobjekt automatisch geschlossen und eventuell eine weitere Verbindung eingegangen. Verbindungsanfragen, die nicht sofort akzeptiert werden, sind keineswegs verloren, sondern werden gepuffert. Sie befinden sich in der sogenannten *Queue* (deutsch, *Warteschlange*) und können nacheinander abgearbeitet werden. Zum Schluss wird der Verbindungssocket mit `close()` geschlossen. Die Struktur des Clients ist vergleichsweise einfach. So gibt es beispielsweise nur einen Kommunikationssocket, über den mithilfe der Methode `connect()` eine Verbindungsanfrage an einen bestimmten Server gesendet werden kann. Danach erfolgt die tatsächliche Kommunikation über `send()` oder `recv()`. Nach dem Ende der Kommunikation wird der Verbindungssocket geschlossen.

3.3 Netzwerkprotokolle

Grundsätzlich kann für die Datenübertragung zwischen Server und Client aus zwei verfügbaren Netzwerkprotokollen gewählt werden: *UDP* und *TCP*. In den folgenden beiden Abschnitten werden kleine Beispielserver und -clients für beide dieser Protokolle implementiert. Beachten Sie, dass sich das obige Flussdiagramm auf das verbindungsbehaftete und üblichere TCP-Protokoll bezieht. Die Handhabung des verbindungslosen UDP-Protokolls unterscheidet sich davon in einigen wesentlichen Punkten. Näheres dazu erfahren Sie im folgenden Abschnitt.

3.3.1 User Datagram Protocol (UDP)

Das Netzwerkprotokoll *UDP (User Datagram Protocol)* wurde 1977 als Alternative zu *TCP (Transmission Control Protocol)* für die Übertragung menschlicher Sprache entwickelt. Charakteristisch ist, dass UDP verbindungslos und nicht zuverlässig ist. Diese beiden Begriffe gehen miteinander einher und bedeuten zum einen, dass keine explizite Verbindung zwischen den Kommunikationspartnern aufgebaut wird, und zum anderen, dass UDP weder garantiert, dass gesendete Pakete in der Reihenfolge ankommen, in der sie gesendet wurden, noch dass sie überhaupt ankommen. Das sind aber nicht unbedingt Nachteile für die Netzwerkkommunikation. Aufgrund dieser Einschränkungen können mit UDP vergleichsweise schnelle Übertragungen stattfinden, da beispielsweise keine Pakete neu angefordert oder gepuffert werden müssen.

Damit eignet sich UDP insbesondere für Multimedia-Anwendungen wie Telefonieren sowie Audio- oder Videostreaming, bei denen es auf eine schnelle Übertragung der Daten ankommt und kleinere Übertragungsfehler toleriert werden können.

Das folgende Beispiel besteht aus einem Server- und einem Clientprogramm. Der Client schickt eine Textnachricht per UDP an eine bestimmte Adresse. Wenn die Nachricht ankommt, nimmt das laufende Serverprogramm die Nachricht entgegen und zeigt sie an. Betrachten Sie zunächst den Quellcode des Clients:

```
1  # Client
2  import socket
3
4  # Erzeugt eine Socket-Instanz mit AF_INET als Adresstyp
5  # 'Internet/IPv4', also eine IP-Adresse, und SOCK_DGRAM
6  # als Netzworkprotokoll, hier UDP
7  s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8
9  # Eingabe von Empfänger-IP und
10 # Nachricht über die Eingabeaufforderung
11 ip = input("IP-Adresse: ")
12 nachricht = input("Nachricht: ")
13
```

die Möglichkeit haben, die Berechnung gegebenenfalls abbrechen zu können. Ein anderes Beispiel ist ein Webserver, der während der Verarbeitung eines Aufrufs der Webseite auch für weitere Zugriffe verfügbar sein muss. Sonst könnte eine Webseite immer nur von einem Besucher gleichzeitig besucht werden.

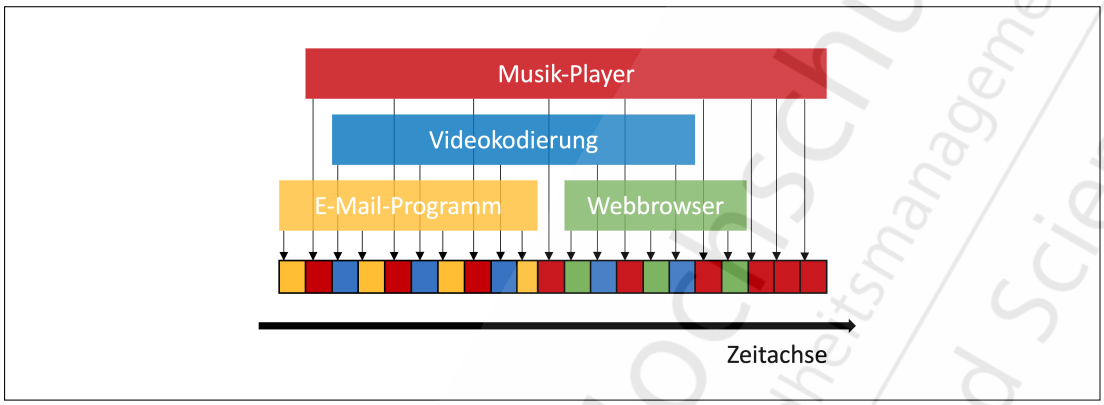


Abbildung 17: Die Prozesse wechseln sich ab und laufen nicht gleichzeitig ab. (©BSA/DHfPG)

Es ist daher möglich, die Beschränkung auf nur eine Operation zur selben Zeit dadurch zu umgehen, dass weitere Prozesse erzeugt werden. Dieses Konzept nennt man auch *Parallelisierung* der Prozesse. Allerdings ist die Erzeugung eines Prozesses recht ressourcenaufwendig. Auch der Datenaustausch zwischen den Prozessen muss geregelt werden, weil jeder Prozess seine eigenen Variablen hat, die von den anderen Prozessen abgeschirmt werden müssen, da sonst Werte in den falschen Operationen auftreten können.

Eine weitere Möglichkeit, um ein Programm zu parallelisieren, bieten sogenannte *Threads*. Ein Thread (deutsch, *Faden*) ist ein Ausführungsstrang innerhalb eines Prozesses und wird häufig auch als „leichtgewichtiger Prozess" bezeichnet. Standardmäßig besitzt jeder Prozess genau einen Thread, der die Ausführung des Prozesses organisiert (siehe Abbildung 18).

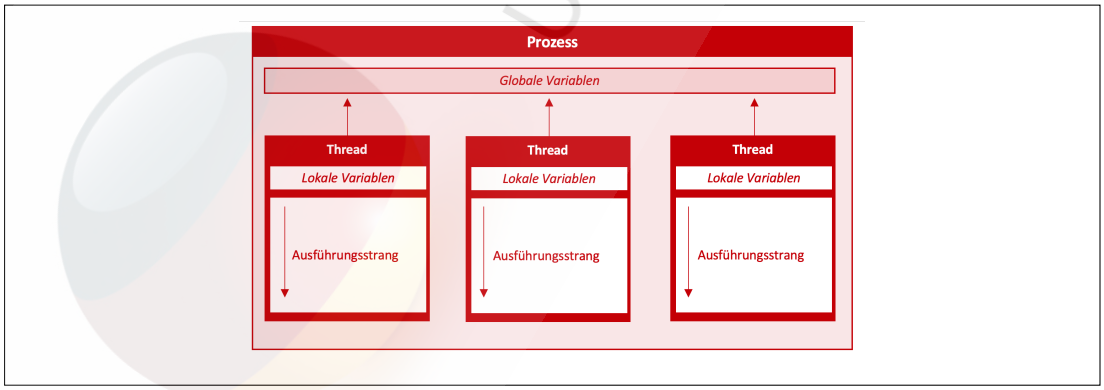


Abbildung 18: Ein Prozess mit drei Threads. Die globalen Variablen des Prozesses sind in allen Threads verfügbar. (©BSA/DHfPG)