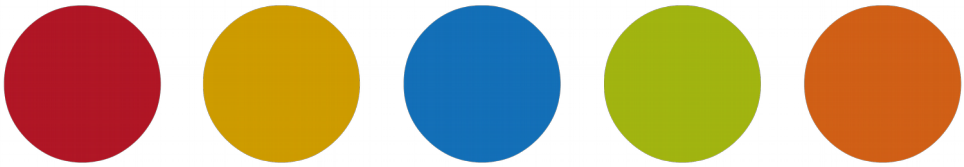
Dokumentation
Algorithmen und Datenstrukturen
in Python II

Skript

Materialsammlung

Schulung:	Informatik und Wirtschaftsinformatik

Stand: 15. Mrz 2021



Urquelle für die Aufgaben und Lösungen ist der Landesbildungsserver BW

überarbeitet und ergänzt:

Christine Janischek
Berufliches Schulzentrum Wangen

Inhaltsverzeichnis

1 Datenstrukturen L1 1.....3
2 Die Datenstruktur Array L1 2.....12
3 Implementierung von Arrays L1 3.....16
4 Grundlagen Algorithmik L2 1.....37
5 Sortieralgorithmen L2 2.....42
6 Suchalgorithmen L2 3.....69
7 Dynamische Datenstrukturen – verkettete Liste L3 1.....91
8 Dynamische Datenstrukturen – Stapelspeicher L3 2.....97
9 Dynamische Datenstrukturen – Warteschlange L3 3.....105
10 Dynamische Datenstrukturen – Baum L3 4.....114

1 Datenstrukturen L1 1

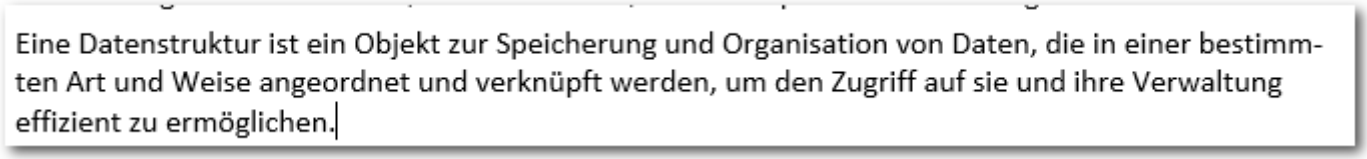
Datenstrukturen L1 1

Thema:	Datenstrukturen und Algorithmen in Python II– Arbeitsauftrag Quelle: L1 1 Datenstrukturen

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Aufgabenstellungen das Informationsmaterial L1_1 Information_Datenstrukturen.docx.

1 Formulieren Sie einen Satz, der den Begriff 'Datenstruktur' definiert.



Eine Datenstruktur ist ein Objekt zur Speicherung und Organisation von Daten, die in einer bestimmten Art und Weise angeordnet und verknüpft werden, um den Zugriff auf sie und ihre Verwaltung effizient zu ermöglichen.

2 Beschreiben Sie die Merkmale folgender Datenstrukturen (Speicherstrukturen):

- 2.1 Array
- 2.2 Verkettete Liste
- 2.3 Stapelspeicher
- 2.4 Warteschlange
- 2.5 Baum

Array:

- Daten: Datenelemente des gleichen Typs
- Einfügen: Zusätzliche Elemente werden hinten angehängt.
- Zugriff: direkter Zugriff über den Index (Adresse: Index-Werte (0, 1, 2, 3, usw.))
- Löschen: Sehr umständlich/aufwendig ((Daten verwalten: hinzufügen, suchen, löschen – aufwendig)
- Struktur: statisch (Größe ist fix!) und dynamische (Größe ist variabel!) Varianten

Beispiel: In einem Krankenhaus wird die Körpertemperatur der Patienten täglich mehrmals gemessen und ausgewertet. Daraus ergibt sich, dass ein *Array* vom Datentyp Dezimal zahl – z.B. temp[ ] – benötigt wird.

38.3	37.9	38.7

Verkettete Liste

- Daten: mehrere Elemente beliebiger Datentypen gespeichert
- Einfügen: Zusätzliche Elemente werden in Listenelementen ist als Knoten → [Element|Zeiger] erfasst. Bei Bedarf können weitere Knoten eingefügt werden*
- Zugriff: Zugriff über Knoten [Element|Zeiger]
- Löschen: *oder entfernt werden. Jeder [Element|Zeiger] ist Knoten enthält einen Verweis auf den nächsten Knoten. Beim Löschen eines Elements muss nur der Zeiger entsprechend festgelegt werden.
- Struktur: dynamische Datenstruktur(Größe ist variabel!)

Beispiel: Bei einem Skisprung-Wettkampf sollen die Haltungsnoten der Sprünge erfasst und ausgewertet werden. Dabei müssen folgende Regeln beachtet werden:

- Jedem Teilnehmer wird ein Versuch gewährt.
- Fünf Sprungrichter vergeben bei jedem Sprung Noten, von denen die beste und die schlechteste gestrichen werden.
- Den acht Springern mit den besten Leistungen steht ein weiterer Sprung zu.

Springer 1: 17,5 / 18,0 / ~~17,0~~ / ~~18,5~~ / 18,0

Springer 2: 19,0 / 19,0 / ~~18,5~~ / 19,5 / ~~20,0~~ / 19,5 / ~~19,0~~ / 19,5 / ~~20,0~~ /19,5

Stapelspeicher (Stack):

- Daten: mehrere Datenelemente/-objekte beliebiger Datentypen gespeichert
- Einfügen: Last-in-First-Out-Prinzip (**LIFO-Prinzip**) - Elemente werden oben auf den Stapel gelegt und auch nur von dort wieder abgerufen
- Zugriff: nur über das oberste (zuletzt eingefügten) Elemente.
- Löschen: LIFO
- Struktur: dynamische Datenstruktur (Größe ist variabel!)

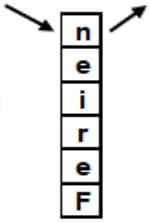
Beispiel: Ein Textverarbeitungsprogramm soll es ermöglichen, das jeweils zuletzt geschriebene Zeichen mit der Backspace-Taste(←) zu löschen.
Da das zu löschende Zeichen immer das zuletzt geschriebene ist, bietet sich die Datenstruktur *Stapelspeicher* an

Geschriebener Text: Ferien|

Löschen des letzten Zeichens.

Eintrag im Stapelspeicher:

letztes eingefügte Element: n
Wird als erstes aus dem Stapel entfernt.



Dieses Prinzip wird auch als Last-in-First-Out-Prinzip (**LIFO**-Prinzip) bezeichnet.

Warteschlange:

- Daten: mehrere Datenelemente/-objekte beliebiger Datentypen gespeichert
- Einfügen: Prinzip wird auch als First-in-First-Out-Prinzip (**FIFO-Prinzip**) - Die Datenobjekte werden immer an das Ende der Warteschlange angefügt
- Zugriff: Rückgabe erfolgt in der Reihenfolge ihres Einfügens. (Daten verwalten: hinzufügen, suchen, löschen)
- Löschen: FIFO - Die Rückgabe der Datenobjekte erfolgt in der Reihenfolge ihres Einfügens
- Struktur: dynamische Datenstruktur (Größe ist variabel!)

Beispiel: Beim Besuch einer Arztpraxis muss jeder Patient eine fortlaufende Nummer ziehen. Die Patienten werden dann in entsprechender Reihenfolge zur Behandlung aufgerufen.
Hier bietet sich die Datenstruktur *Warteschlange* an, da der am längsten wartende Patient als erstes an die Reihe kommt. Seine Nummer steht am Anfang der Warteschlange.

1. Patient letzter Patient

133	134	135	136	137

Als erstes wird der Patient mit der Nummer 133 aufgerufen.

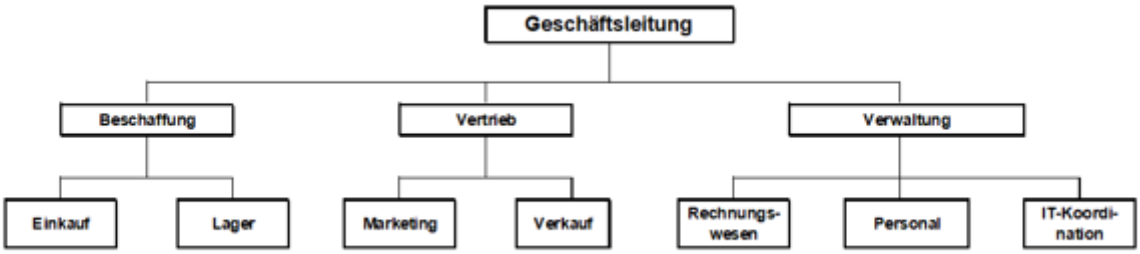
Dieses Prinzip wird auch als First-in-First-Out-Prinzip (**FIFO**-Prinzip) bezeichnet.

Baum:

- Daten: mehrere unterschiedliche Datenobjekte (Knoten) hierarchisch gespeichert (root/Wurzel-Element mit Elternelemente → verweisende Knoten auf Kindelemente → untergeordnete Knoten), Abbildung hierarchischer Strukturen (→ Ein Vorfahre)
- Einfügen: Knoten
- Zugriff: Über Knoten.
- Löschen: Knoten
- Struktur: dynamische Datenstruktur (Größe ist variabel!)

z. B. Dateiverwaltung, Organisationen

Beispiel: Die Holzwurm GmbH, möchte ihre Unternehmensstruktur digital erfassen.
Der Geschäftsleitung sind die Abteilungen Beschaffung, Vertrieb und Verwaltung direkt unterstellt. Die Beschaffung gliedert sich in die Bereiche Einkauf und Lagerhaltung. Der Vertrieb in Marketing und Verkauf, die Verwaltung in Rechnungswesen, Personal und IT-Koordination.
Hier bietet sich die Datenstruktur *Baum* an, da die zu erfassenden Daten eine hierarchische, baumartige Beziehung untereinander haben.



3 Begründen Sie für die folgenden Sachverhalte, welche Datenstruktur jeweils zu wählen ist.

3.1 Bei einem physikalischen Experiment zum Thema 'Freier Fall' werden Metallmuttern in bestimmten Abständen an eine Schnur gebunden, deren unteres Ende den Boden berührt. Die Schnur mit den Metallmuttern wird fallen gelassen und die jeweilige Zeit des Auftreffens der Muttern auf den Boden gemessen und erfasst.

Die Muttern befinden sich im ersten Versuch in den Abständen 5cm, 10cm, 20cm, 40cm, 80cm, 160cm (gemessen vom Boden).

Im zweiten Versuch in den Abständen 5cm, 10cm, 20cm, 40cm, 45cm, 80cm, 125cm, 160cm und 180cm.

Im dritten Versuch in den Abständen 5cm, 20cm, 45cm, 80cm, 125cm und 180cm.

Verkettete Liste:
- mehrere Elemente beliebiger Datentypen (Abstände – in cm und Auftreffen – in Sekunden)
- leichtes löschen von Elementen (Zeiger neu referenzieren – Daten verwalten: hinzufügen, suchen, löschen)
- Daten werden in den Listenelementen, den sogenannten Konten erfasst.
- Jeder Knoten enthält einen Verweis zum nächsten Knoten, dem sogenannten Zeiger.

3.2 Das Ergebnis der Ziehung der Lottozahlen soll digital erfasst werden. Die Ziehung der Superzahl soll dabei nicht berücksichtigt werden.

Array:
- Datenelemente des gleichen Datentyps
- zusätzliche Elemente werden hinten angehängt (dyn.Array = variable Anzahl)
- direkter Zugriff über Index-Werte (0, 1, 2, 3, usw.)
- Elemente löschen ist aufwendig. (Daten verwalten: hinzufügen, suchen, löschen - aufwendig)

3.3 Der Sportverein Südstadt e.V. plant einen neuen Internetauftritt. Die Struktur des Seitenaufbaus finden Sie in der Anlage (Folgeseite).

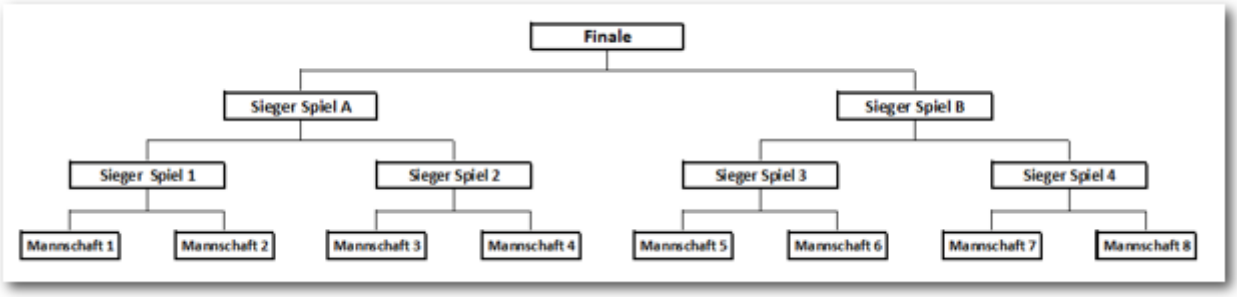
Die einzelnen Seiten können mit den Navigationslinks aufgerufen werden. Mit Hilfe der zurück-Links sollen die jeweils zuvor aufgerufenen Seiten anwählbar sein. Dazu müssen die jeweiligen Seitennamen (startseite.html, fussbal_1.html etc.) gespeichert werden.

Zwei Datenstrukturen verwendet!

Stapelspeicher (Stack): → Zurück Funktion
- Elemente werden oben auf den Stapel gelegt und auch nur von dort wieder abgerufen werden.
- Dieses Prinzip wird auch als Last-in-First-Out-Prinzip (**LIFO-Prinzip**) bezeichnet. (Daten verwalten: hinzufügen, suchen, löschen)
- Dyn. Datenstruktur

Baum: → Seitenverwaltung (Startseite (root), Unterseiten, Unter-Unterseiten)
- Abbildung hierarchischer Strukturen
- Datenelemente werden in Knoten erfasst
- Hierarchie Elternknoten (verweisende Knoten), Kinder (untergeordnete Knoten)
- Ein Vorfahre (Wurzel, Root) → z. B. Dateiverwaltung
  → Startseite index.html des Webprojekts
- Dyn. Datenstruktur

3.4 Der Sportverein Südstadt e.V. veranstaltet ein Fußballturnier mit acht beteiligten Mannschaften. Das Turnier soll im K.O.-Modus stattfinden. Der Spiel plan des Turniers soll digital erfasst werden.



Baum: → Hierachieebenen (Finale (root), Sieger und Mannschaften)

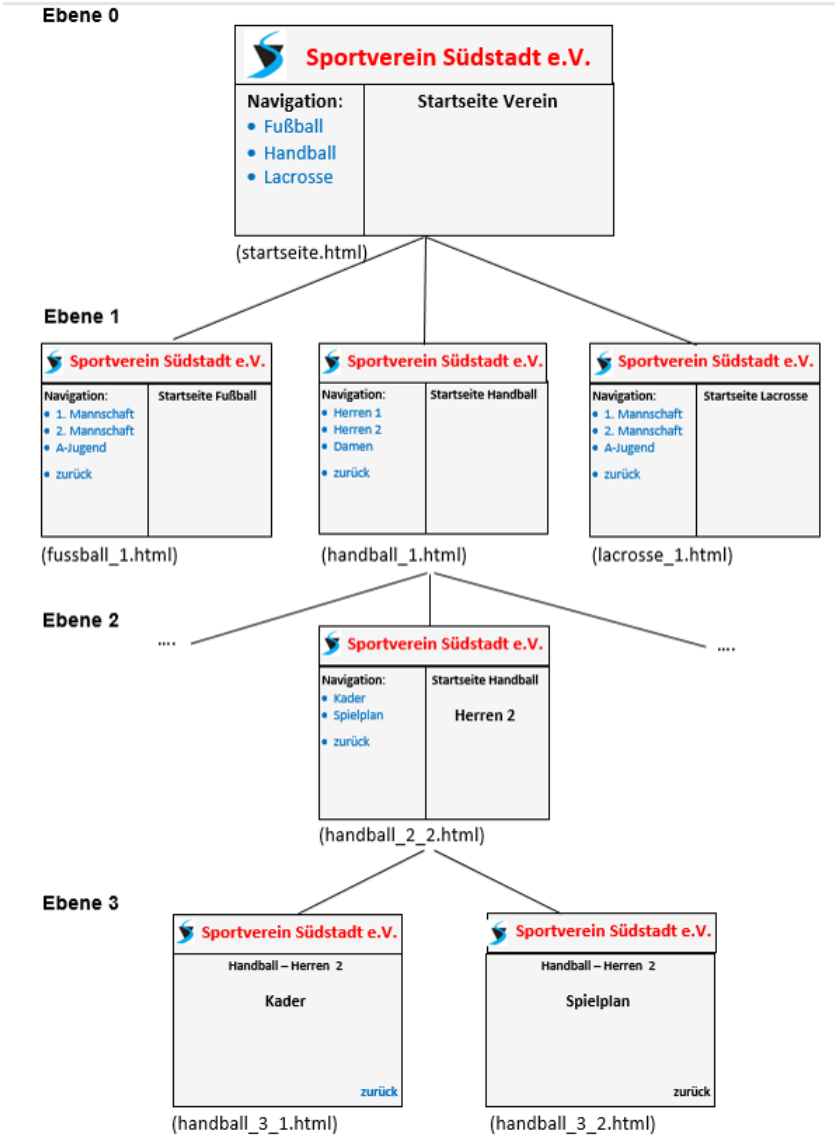
- Abbildung hierarchischer Strukturen
- Datenelemente werden in Knoten erfasst
- Hierarchie Elternknoten (verweisende Knoten), Kinder (untergeordnete Knoten)
- Ein Vorfahre (Wurzel, Root) → z. B. Dateiverwaltung
  → Finale
- Dyn. Datenstruktur

3.5 Zur Kontrolle der Verkehrssicherheit im Elbtunnel (Hamburg) sollen die Kenn zeichen aller Fahrzeuge, die in den Tunnel einfahren, digital gespeichert wer den. Nach der Ausfahrt aus dem Tunnel werden die gespeicherten Kennzeichen sofort wieder gelöscht. Es wird unterstellt, dass der Tunnel einspurig ist und ein Überholverbot gilt.

Warteschlange:

- Die Datenobjekte werden immer an das Ende der Warteschlange angefügt
- Die Rückgabe der Datenobjekte erfolgt in der Reihenfolge ihres Einfügens
- Dieses Prinzip wird auch als First-in-First-Out-Prinzip (**FIFO-Prinzip**) bezeichnet. (Daten verwalten: hinzufügen, suchen, löschen) → Einfahren und Ausfahren
- Dyn. Datenstruktur

Anlage: Struktur der Internetpräsenz des Sportvereins Südstadt e.V.



2 Die Datenstruktur Array L1 2

Datenstrukturen L1 1

Thema:	Datenstrukturen und Algorithmen in Python II– Arbeitsauftrag Quelle: L1 2 Array

L1 2 Aufgabe Array

Aufgaben:

1 Formulieren Sie den Code, der die nachfolgend beschriebenen Anweisungen ausführt.

1.1 Erzeugen Sie ein Array mit dem Namen weltmeister und den Werten 1954, 1974, 1990 und 2014.

1.2 Erzeugen Sie ein Array wmtrainer und den Werten „Sepp Herberger", „Helmut Schön" und „Franz Beckenbauer":

1.3 Fügen Sie in die Liste wmtrainer den Namen „Joachim Löw" am Ende ein:

1.4 Im Array weltmeister sind die Jahre gespeichert, in denen die deutsche Fußball-Nationalmannschaft Weltmeister geworden ist. Die Anzahl der Weltmeistertitel soll in der Konsole ausgegeben werden.

1.5 Alle im Array wmtrainer gespeicherten Namen sollen in der Konsole angezeigt werden.

LÖSUNG:

L1_2_Array_Weltmeister.py

```
#EINGABE: Deklarieren und Initialisierung des Arrays (ints)
weltmeister = [1954,1974,1990,2014]

#EINGABE: Deklarieren und Initialisierung des Arrays (Strings)
wmtrainer = ["Sepp Herberger","Helmut Schön","Franz Beckenbauer"]

#VERARBEITUNG: Hinzufügen/Einfügen in diesem Anhängen eines neuen Elements
wmtrainer.append("Joachim Löw")

#VERARBEITUNG: Funktion zur Ermittlung und
#Ausgabe der Anzahl an Elementen eines Arrays
def weltmeister_anzahl_ausgeben(weltmeister):
    return len(weltmeister)

#VERARBEITUNG: Funktion/Methode zur Ausgabe eines Arrays
def wmtrainer_ausgeben(wmtrainer,anzahl):
    for i in range(anzahl):
        print(wmtrainer[i])

#VERARBEITUNG: Funktions-/Methodenaufrufe
laenge = weltmeister_anzahl_ausgeben(weltmeister)
anzahl = len(wmtrainer)

#AUSGABE:
print("Länge/Anzahl der Elemente im Weltmeister-Array:", laenge)
print("-------------------------------------------------------")
print("Liste der WM-Trainer:")
print("------------------------------")
wmtrainer_ausgeben(wmtrainer,anzahl)
```

```
Länge/Anzahl der Elemente im Weltmeister-Array: 4
-------------------------------------------------------
Liste der WM-Trainer:
------------------------------
Sepp Herberger
Helmut Schön
Franz Beckenbauer
Joachim Löw
```

2 Das Array a = [1, 5, 20, 9, 4, 3, 1] ist gegeben und wird durch den folgenden Algorithmus verändert. Durchlaufen Sie den Algorithmus mit dem gegebenen Array und nennen Sie den Arrayinhalt nach jeder Anweisung. (Diese Aufgabe kann hier auf Papier gelöst werden.)

Algorithmus	Array
a = [1, 5, 20, 9, 4, 3, 1]	[1, 5, 20, 9, 4, 3, 1]
a[1] = a[2]	
a[5] = a[5] + a[6]	
a[6] = 4	
a.append(4)	
i = 3	
a[i] = i	

LÖSUNG:

Python_II_L2_2_1_1_BubbleSort_Lottozahlen.py | Python_II_L1_3_Array_Schreibtischtest.py

```
1  import array
2
3  #EINGABE: Deklaration und Initialisierung der Array -> i
4  a = [1, 5, 20, 9, 4, 3, 1]
5
6
7  #VERARBEITUNG: Funktionen zur schrittweise Verarbeitung
8
9  def schreibtischtest_ausgeben(a):
10     #Ausgabe der Werte
11     anweisung1 = print(a)
12
13     a[1] = a[2]
14     anweisung2 = print(a)
15
16     a[5] = a[5] + a[6]
17     anweisung3 = print(a)
18
19     a[6] = 4
20     anweisung4 = print(a)
21
22     a.append(4)
23     anweisung4 = print(a)
24
25     i = 3
26     a[i] = i
27     anweisung5 = print(a)
28
29
30 #AUSGABE
31 print("Schreibtischtest:")
32 print("-----------------")
33 #VERARBEITUNG: Methodenaufruf für die ausgabe
34 schreibtischtest_ausgeben(a)
```

Shell

```
>>> %Run Python_II_L1_3_Array_
  Schreibtischtest:
  -----------------
  [1, 5, 20, 9, 4, 3, 1]
  [1, 20, 20, 9, 4, 3, 1]
  [1, 20, 20, 9, 4, 4, 1]
  [1, 20, 20, 9, 4, 4, 4]
  [1, 20, 20, 9, 4, 4, 4, 4]
  [1, 20, 20, 3, 4, 4, 4, 4]
```

3 Implementierung von Arrays L1 3

Implementierung von Arrays L1 3

Thema:	Datenstrukturen und Algorithmen in Python II– Arbeitsauftrag Quelle: L1 3 1 1 Arbeitsauftrag Vereinsmeisterschaften

Ausgabe von Arrays – Vereinsmeisterschaften

Der Mühlberger SC ist ein Sportverein mit mehreren Abteilungen. Im Jahr 2016 eröffnete der Mühlberger SC die Abteilung DART. Seit dem Jahr 2017 finden auch Vereinsmeisterschaften statt.

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Problemstellung das Informations-material L1_2 Information Array.docx.

(I) Problemstellung

Die Vereinsmeisterschaft in diesem Jahr war ein großer Erfolg. Die Spieler lieferten sich einen spannenden Wettkampf vor vielen Zuschauern. Der Mühlberger SC möchte die Ergebnisse des Wettkampfs veröffentlichen. Nutzer sollen die Möglichkeit haben, sich an einem Computer eine Platzierung ausgeben zu lassen. Möchte der Nutzer den Erstplatzierten ausgegeben bekommen, muss er bspw. eine „1" eingeben. Außerdem soll die Teilnehmerzahl angezeigt werden. Die Teilnehmer an der Meisterschaft sind in dem Array platzierungen in der Reihenfolge ihrer Platzierung gespeichert.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen

L1_3_1_1_vereinsmeisterschaften.py.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

Name des Teilnehmers auf deinem bestimmten Platz, Teilnehmeranzahl

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Ziffer der Platzierung

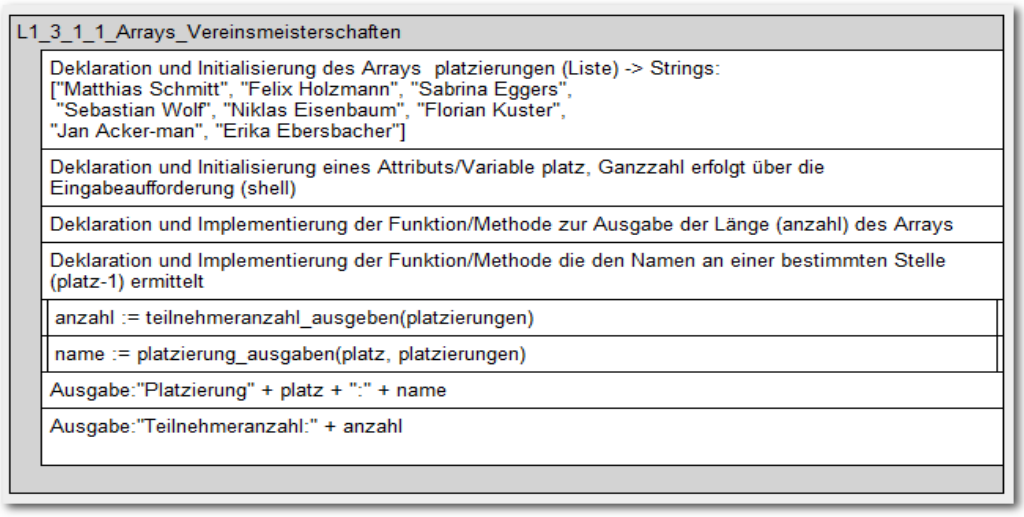
(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten? (Variablenliste)

Bedeutung	Typ/Struktur	Variable/Größe
Teilnehmer (gegeben)	Array platzierung	
Teilnehmeranzahl	Ganzzahl	len(platzierung)

(4) Gewünschter Ablauf des Programms mit Beispieldaten:

Eingabe	`Welche Platzierung soll ausgegeben werden? 2`
Ausgabe	`Platzierung 2: Felix Holzmann` `Teilnehmerzahl: 8`

(III) Struktogramm



(IV) Programmcode (Python-Code)

Python_II_L1_3_Array_Schreibtischtest.py | Python_II_L1_3_1_1_Array_vereinsmeisterschaften.py

```python
#EINGABE: Deklaration und Initialisierung der Array -> Strings:
platzierungen = ["Matthias Schmitt", "Felix Holzmann", "Sabrina Eggers",
                 "Sebastian Wolf", "Niklas Eisenbaum", "Florian Kuster",
                 "Jan Ackermann", "Erika Ebersbacher"]


#EINGABE: Deklaration und Initialisierung über die Eingabeaufforderung
platz = int(input("Welche Platzierung soll ausgegeben werden (1-8)?"))

#VERARBEITUNG: Funktion zur Ausgabe der Länge des Arrays (Anzahl)
def teilnehmeranzahl_ausgeben(platzierungen):
    return len(platzierungen)

#VERARBEITUNG: Funktionen zur Ausgabe eines Arrays mit einer Schleife
def platzierung_ausgeben(platzierungen,platz):
    teilnehmer = platzierungen[platz-1]
    return teilnehmer

#VERARBEITUNG: Methodenaufrufe
anzahl = teilnehmeranzahl_ausgeben(platzierungen)
name = platzierung_ausgeben(platzierungen,platz)

#AUSGABE
print("Platzierung:", name)
print("---------------------------")
print("Anzahl der Teilnehmer:", anzahl)
print("---------------------------")

```

```
>>> %Run Python_II_L1_3_1_1_Array_vereinsmeisterschaften.py
  Welche Platzierung soll ausgegeben werden (1-8)?1
  Platzierung: Matthias Schmitt
  ---------------------------
  Anzahl der Teilnehmer: 8
  ---------------------------
```

Ausgabe

Thema:	Datenstrukturen und Algorithmen in Python II– Arbeitsauftrag Quelle: L1 3 1 2 Arbeitsauftrag Volleyball – Spieler anzeigen

L1_3.1.2 Volleyball: Spieler anzeigen

(I) Problemstellung

Die Abteilung Volleyball des Sportvereines Mühlbeger SC besteht zurzeit aus 12 Spielern. Die Namen der Stammspieler sind in dem Array *spieler* in der Reihenfolge ihrer Startpositionen erfasst.

```
spieler = ["Armin", "Batu", "Kai", "Sven", "Paul", "Milan"];
```

Die Ersatzspieler sind im Array *ersatz* in alphabetischer Reihenfolge erfasst.

```
ersatz = ["Chris", "Dennis", "Emin", "Goran", "Luca", "Nico"];
```

Der Trainer der Mannschaft möchte eine Software, mit deren Hilfe er sich die Namen der Stammspieler, die der Ersatzspieler und die des gesamten Kaders anzeigen lassen kann. Der Co-Trainier hat bereits mit der Implementierung der Eingabe begonnen (siehe Vorlage). Es steht nur noch die Implementierung der Funktionen aus.

Implementieren Sie

- eine Funktion `zeige_startaufstellung()`, mit der die Namen der Stammspieler
- eine Funktion `zeige_ersatzspieler()`, mit der die Namen der Ersatzspieler
- eine Funktion `zeige_kader()`, mit der die Namen aller Spieler im Kader der Mannschaft

in der Konsole angezeigt werden (siehe Folgeseite: (4) Gewünschter Ablauf des Programms).

> Hinweis: Die Inhalte mehrerer Arrays lassen mit folgender Anweisung einfach zu einem Array zusammenfügen:
>
> ```
> arrayNeu = array1 + array2
> ```

Optional:

Verwenden Sie für die Implementierung Ihrer Lösung die Datei *L1_3_1_2_Vorlage_volleyball_spieler_anzeigen.py*, die Ihnen im Ordner *Vorlagen* in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen *L1_3_1_2_volleyball_spieler_anzeigen.py*.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

Elemente des Arrays spieler[]

Elemente des Arrays ersatz[]

Elemente des Arrays kader[]

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Anzeigewunsch

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten? (**Variablenliste**)

Bedeutung	Typ/Struktur	Variable/Größe
Elemente des Arrays spieler[]	Array	spieler
Elemente des Arrays ersatz[]	Array	ersatz
Elemente des Arrays kader[]	Array	kader
Anzeigewunsch	Ganzzahl	anzeige

(4) Gewünschter Ablauf des Programms:

<table>
<tr><td rowspan="2">Eingabe</td><td colspan="3">(1) Startaufstellung anzeigen
(2) Ersatzspieler anzeigen

(3) Kader anzeigen</td></tr>
<tr><td>Anzeigewunsch (1-3):
1</td><td>Anzeigewunsch (1-3):
2</td><td>Anzeigewunsch (1-3):
3</td></tr>
<tr><td>Ausgabe</td><td>----------
-
Startaufstellung

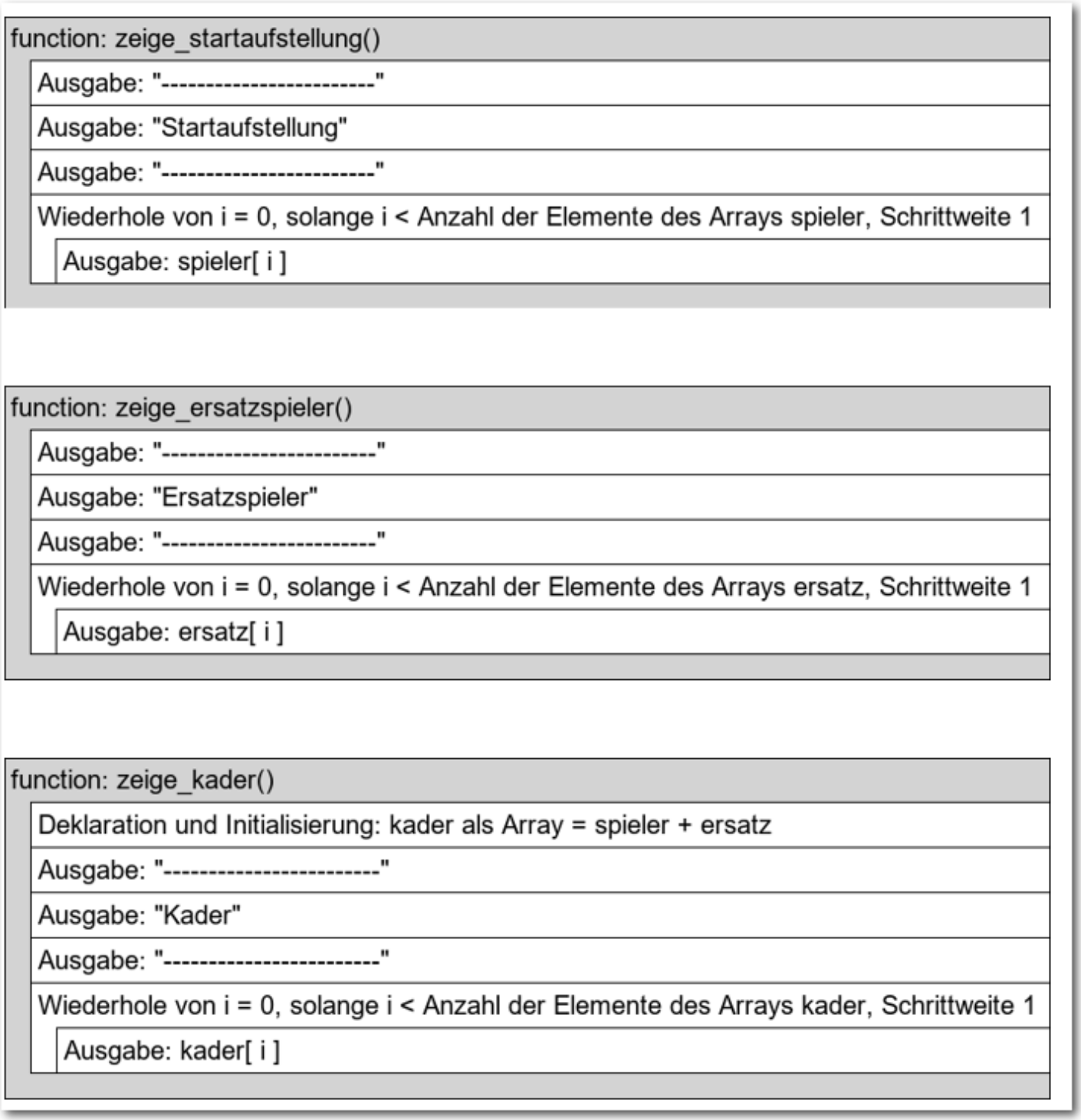
-
Armin
Batu
Kai
Sven
Paul
Milan</td><td>----------
-
Ersatzspieler

-
Chris
Dennis
Emin
Goran
Luca
Nico</td><td>----------
-
Kader

-
Armin
Batu
Kai
Sven
Paul
Milan
Chris
Dennis
Emin
Goran
Luca
Nico</td></tr>
</table>

(III) Struktogramm

Die Arrays *spieler* und *ersatz* sind bereits implementiert!



(IV) Programmcode (Python-Code)

Python_II_L1_3_1_2_Array_Volleyball_anzeigen.py

```
#EINGABE: Deklaration und Initialisierung der Array -> Strings:
spieler = ["Armin", "Batu", "Kai", "Sven", "Paul", "Milan"]
ersatz = ["Chris", "Dennis", "Emin", "Goran", "Luca", "Nico"]
kader = spieler + ersatz

#VERARBEITUNG: Funktion
def menue_anzeigen():
    print("(1) Startaufstellung anzeigen")
    print("(2) Ersatzspieler anzeigen")
    print("(3) Kader anzeigen")
#VERARBEITUNG: Funktion
def zeige_startaufstellung():
    print("--------------------")
    print("Startaufstellung")
    print("--------------------")
    i = 0
    while i < len(spieler):
        print(spieler[i])
        i = i+1
#VERARBEITUNG: Funktionen
def zeige_ersatzspieler():
    print("--------------------")
    print("Ersatzspieler")
    print("--------------------")
    i = 0
    while i < len(ersatz):
        print(ersatz[i])
        i = i+1
#VERARBEITUNG: Funktionen
def zeige_kader():
    print("--------------------")
    print("Kader")
    print("--------------------")
    i = 0
    while i < len(kader):
        print(kader[i])
        i = i+1

#VERARBEITUNG + AUSGABE: Methodenaufrufe
def auswahl_anzeigen(anzeigewunsch):
    if(anzeigewunsch == 1):
        zeige_startaufstellung()
    elif(anzeigewunsch == 2):
        zeige_ersatzspieler()
    elif(anzeigewunsch == 3):
        zeige_kader()

#AUSGABE
menue_anzeigen()
#EINGABE: Deklaration und Initialisierung über d
anzeigewunsch = int(input("Anzeigewunsch (1-3):"
#Methodenaufruf
auswahl_anzeigen(anzeigewunsch)

```

Shell

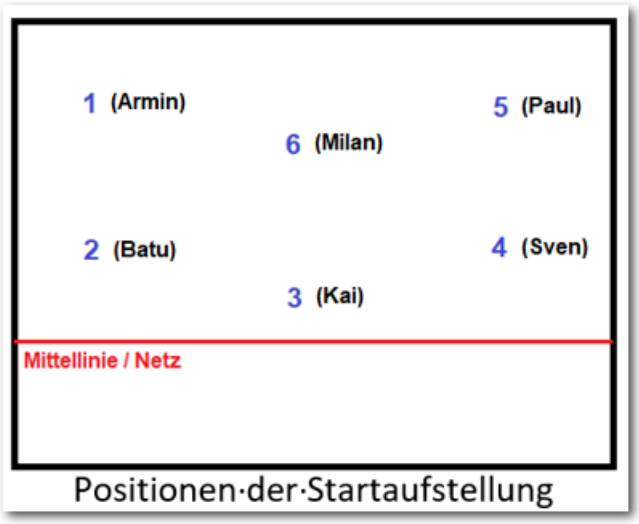
```
>>> %Run Python_II_L1_3_1_2_Array_Volleyball_anzeigen.py

   (1) Startaufstellung anzeigen
   (2) Ersatzspieler anzeigen
   (3) Kader anzeigen
   Anzeigewunsch (1-3):2
   --------------------
   Ersatzspieler
   --------------------
   Chris
   Dennis
   Emin
   Goran
   Luca
   Nico

>>> %Run Python_II_L1_3_1_2_Array_Volleyball_anzeigen.py

   (1) Startaufstellung anzeigen
   (2) Ersatzspieler anzeigen
   (3) Kader anzeigen
   Anzeigewunsch (1-3):3
   --------------------
   Kader
   --------------------
   Armin
   Batu
   Kai
   Sven
   Paul
   Milan
   Chris
   Dennis
   Emin
   Goran
   Luca
   Nico

>>> %Run Python_II_L1_3_1_2_Array_Volleyball_anzeigen.py

   (1) Startaufstellung anzeigen
   (2) Ersatzspieler anzeigen
   (3) Kader anzeigen
   Anzeigewunsch (1-3):1
   --------------------
   Startaufstellung
   --------------------
   Armin
   Batu
   Kai
   Sven
   Paul
   Milan

>>> %Run Python_II_L1_3_1_2_Array_Volleyball_anzeigen.py

   (1) Startaufstellung anzeigen
   (2) Ersatzspieler anzeigen
   (3) Kader anzeigen
   Anzeigewunsch (1-3):2
   --------------------
   Ersatzspieler
   --------------------
   Chris
   Dennis
   Emin
   Goran
   Luca
   Nico

>>> %Run Python_II_L1_3_1_2_Array_Volleyball_anzeigen.py

   (1) Startaufstellung anzeigen
   (2) Ersatzspieler anzeigen
   (3) Kader anzeigen
   Anzeigewunsch (1-3):3
   --------------------
   Kader
   --------------------
   Armin
   Batu
   Kai
   Sven
   Paul
   Milan
   Chris
   Dennis
   Emin
   Goran
   Luca
   Nico
```

Thema:	Datenstrukturen und Algorithmen in Python II – Arbeitsauftrag Quelle: L1 3 4 Arbeitsauftrag Volleyball Positionen tauschen

L1_3.4 Volleyball: Spielerpositionen tauschen – Index

(I) Problemstellung

Der Trainer der Abteilung Volleyball des Sportvereines Mühlberger SC möchte eine Erweiterung seiner Software.
Die Namen der Stammspieler sind in dem Array spieler in der Reihenfolge ihrer Startpositionen erfasst.
spieler = ["Armin", "Batu", "Kai", "Sven", "Paul", "Milan"]



Positionen·der·Startaufstellung

Beachten Sie:
Die Startposition 1 erhält den Index 0 des Arrays.

Die Software soll es ermöglichen, Spielerpositionen in der Startaufstellung zu tauschen. Nach der Eingabe von zwei Spielerpositionen sollen die entsprechenden Spielernamen im Array spieler getauscht und anschließend das Array mit der neuen Startaufstellung angezeigt werden. (Die Funktion zeige_startaufstellung() ist bereits implementiert!)

Z.B.: Tausch der Spielerpositionen 2 (Batu) und 4 (Sven).
Das Array spieler soll danach folgenden Inhalt haben:
spieler = ["Armin", "Sven", "Kai", "Batu", "Paul", "Milan"]

Implementieren Sie eine Funktion mannschaft_umstellen_pos(), die das beschriebene Problem löst.

Optional:
Verwenden Sie für die Implementierung Ihrer Lösung die Datei L1_3_4_vorlage_volleyball_positionen_tauschen_index.py, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen L1_3_4_volleyball_positionen_tauschen_index.html.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

Elemente des Arrays spieler

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Erste zu tauschende Spielerposition Zweite zu tauschende Spielerposition

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten? (Variablenliste)

Bedeutung	Typ/Struktur	Variable/Größe
Nummer der ersten Position	Ganzzahl	position_a
Nummer der zweiten Position	Ganzzahl	position_b
Zwischenspeicher	Zeichenkette	zwischenspeicher

(4) Gewünschter Ablauf des Programms mit Beispieldaten:

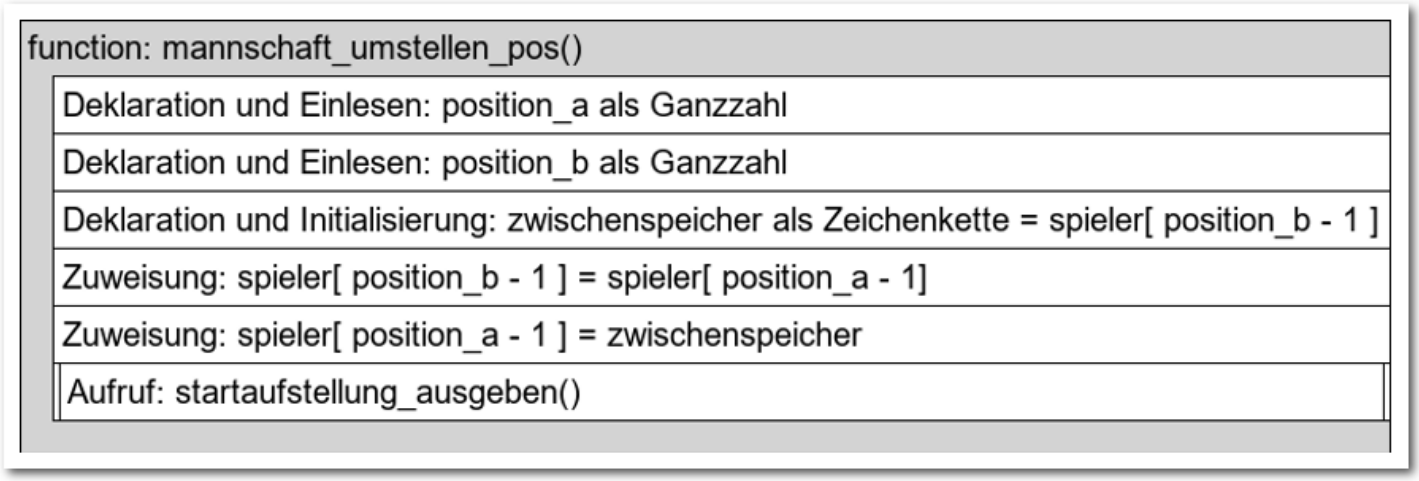
Funktion mannschaft_umstellen_pos()	
Eingabe	Startaufstellung umstellen Tausche Position: 2 mit Position: 4
Ausgabe	---------- Neue Startaufstellung ---------- Armin Sven Kai Batu Paul Milan

(5) Verarbeitung

Die Variable zwischenspeicher erhält den Eintrag der Stelle position_b – 1 des Arrays spieler. Das Array spieler erhält an der Stelle position_b – 1 den Eintrag aus der Stelle position_a -1. Das Array spieler erhält an der Stelle position_a – 1 den Wert der Variablen zwischenspeicher. Die Funktion zeigeAufstellung() wird aufgerufen.

(III) Struktogramm

Das Array spieler und die Funktion zeige_startaufstellung() sind bereits implementiert!



(IV) Programmcode (Python-Code)

Python_II_L1_3_4_Array_Volleyball_tauschen.py

```
52  def mannschaft_umstellen_pos():
53      print("----------------------")
54      print("Startaufstellung umstellen")
55      position_von = int(input("Tausche Position:"))
56      position_nach = int(input("mit Position:"))
57
58      #Tauschalgorithmus
59      for i in range(len(spieler)):
60          if(i == (position_von - 1)):
61              zwischenspeicher = spieler[i]
62              spieler[i] = spieler[position_nach-1]
63              spieler[position_nach - 1] = zwischenspeicher
64
65      #Ausgabe
66      print("----------------------")
67      print("Neue Startaufstellung")
68      print("----------------------")
69      for i in range(len(spieler)):
70           print(spieler[i])
71
72
73  #AUSGABE
74  menue_anzeigen()
75  #EINGABE: Deklaration und Initialisierung über die Eingabeaufforderung
76  anzeigewunsch = int(input("Anzeigewu
77  #Methodenaufruf
78  auswahl_anzeigen(anzeigewunsch)
79
80  #L1_3_4 Array Volleballspieler tausc
81  mannschaft_umstellen_pos()
82
```

Shell

```
>>> %Run Python_II_L1_3_4_Array_Volleyball_tauschen.py
 (1) Startaufstellung anzeigen
 (2) Ersatzspieler anzeigen
 (3) Kader anzeigen
Anzeigewunsch (1-3):1
----------------------
Startaufstellung
----------------------
Armin
Batu
Kai
Sven
Paul
Milan
----------------------
Startaufstellung umstellen
Tausche Position:2
mit Position:4
----------------------
Neue Startaufstellung
----------------------
Armin
Sven
Kai
Batu
Paul
Milan
>>>
```

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L1 3 5 Arbeitsauftrag Volleyball Inhalte einfügen

L1_3.5 Volleyball – Inhalte einfügen

Hinweis: Beachten Sie zur Bearbeitung der nachfolgenden Problemtellung das Informationsmaterial

L1_3.5 Information Elemente einfügen.docx

(I) Problemstellung

Die Software des Trainers der Abteilung Volleyball des Sportvereines Mühlberger SC enthält ein Array mit allen Spielernamen des Mannschaftskaders.

kader = ["Armin", "Batu", "Kai", "Sven", "Paul", "Milan", "Goran", "Chris", "Nico",

"Dennis", "Emin", "Luca"]

Mit Hilfe der Software soll es möglich sein, an einer bestimmten Stelle einen weiteren Spielernamen einzutragen, z.B. an der Stelle mit dem Index 6.

Implementieren Sie eine Funktion einfuegen_spieler(), mit der ein Spieler an einer vom Anwender einzugebenden Stelle, in das Array eingefügt wird.

z. B.: Spielername: Tom - Index: 6

Das Array kader soll danach folgenden Inhalt haben:

["Armin", "Batu", "Kai", "Sven", "Paul", "Milan", "Tom", "Goran", "Chris", "Nico",

"Dennis", "Emin", "Luca"]

Verwenden Sie für die Implementierung Ihrer Lösung die Datei L1_3_5_vorlage_volleyball_spieler

_einfuegen.py, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen L1_3_5_volleyball_spieler_einfuegen.py.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

Array mit erweitertem Eintrag

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Index der Position, an welcher der neue Eintrag erfolgen soll

Text des Eintrags

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten? (Variablenliste)

Bedeutung	**Typ/Struktur**	**Variable/Größe**
Anzahl der Elemente des Arrays *kader*	Ganzzahl	laenge
Index des neuen Eintrags	Ganzzahl	index
Name des neuen Spielers	Zeichenkette	spielername

(4) Gewünschter Ablauf des Programms mit Beispieldaten:

Funktion **einfuegen_spieler()**	
Eingabe	Spielername: Tom Index: 6
Ausgabe	---------- Kader ---------- Armin Batu Kai Sven Paul Milan Tom Goran Chris Nico Dennis Emin Luca

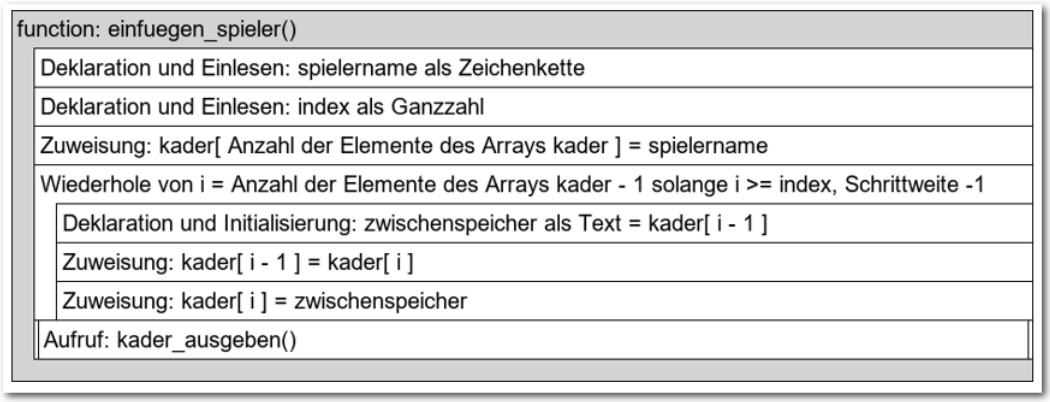
(5) Verarbeitung

- Füge Spielername am Ende des Arrays kader ein.
- Tausche den eingefügten Eintrag am Ende des Arrays kader

(Index: len(kader) - 1) bis zur gewünschten Stelle (index) nach vorne.

(III) Struktogramm

Das Array kader und die Funktion zeige_kader() sind bereits implementiert!



(IV) Programmcode (Python-Code)

Python_II_L1_3_5_Array_Volleyball_einfuegen.py

```
74
75  #VERARBEITUNG: Funktion
76  def einfuegen_spieler():
77      print("----------------------")
78      #EINGABE
79      spielername = input("Spielername:")
80      index = int(input("Index:"))
81
82      #spielername wird einfach angehängt
83      kader.append(spielername)
84      i = len(kader) - 1
85
86      #VERARBEITUNG EINFÜGE-Algorithmus
87      #neues Element wird schrittweise von
88      #hinten nach vorne an die richtige Stelle (Index) gerückt
89      #alternativ: for i in range(i,index, -1):
90      while i >= index:
91          parke = kader[i-1]
92          kader[i-1] = kader[i]
93          kader[i] = parke
94          i = i - 1
95
96      #AUSGABE
97      zeige_kader()
98
99
100 #AUSGABE
101 menue_anzeigen()
102 #EINGABE: Deklaration und Initialisierung
103 anzeigewunsch = int(input("Anzeigewunsch (
104 #Methodenaufruf
105 auswahl_anzeigen(anzeigewunsch)
106
107 #L1_3_4 Array Volleballspieler tauschen
108 #mannschaft_umstellen_pos()
109
110 #L1_3_4 Array Volleballspieler einfügen
111 einfuegen_spieler()
112
```

Shell

```
>>> %Run Python_II_L1_3_5_Array_Volleyball_einfuegen.py
  (1) Startaufstellung anzeigen
  (2) Ersatzspieler anzeigen
  (3) Kader anzeigen
  Anzeigewunsch (1-3):3
  ----------------------
  Kader
  ----------------------
  Armin
  Batu
  Kai
  Sven
  Paul
  Milan
  Goran
  Chris
  Nico
  Dennis
  Emin
  Luca
  ----------------------
  Spielername:Tom
  Index:6
  ----------------------
  Kader
  ----------------------
  Armin
  Batu
  Kai
  Sven
  Paul
  Tom
  Milan
  Goran
  Chris
  Nico
  Dennis
  Emin
  Luca

>>>
```

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L1 3 6 Arbeitsauftrag Volleyball Inhalte entfernen

L1_3.6 Volleyball – Inhalte entfernen

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Problemstellung das Informationsmaterial L1_3.6 Information_Elemente_entfernen.docx

(I) Problemstellung

Die Software des Trainers der Abteilung Volleyball des Sportvereines Mühlberger SC enthält ein Array mit allen Spielernamen des Mannschaftskaders.

kader = ["Armin", "Batu", "Kai", "Sven", "Paul", "Milan", "Goran", "Chris", "Nico",
"Dennis", "Emin", "Luca"];

Mit Hilfe der Software soll es möglich sein, den Spieler an einer bestimmten Stelle des Arrays zu entfernen.

z. B.: Spieler entfernen an der Stelle mit dem Index: 4

Das Array kader soll danach folgenden Inhalt haben:

kader = ["Armin", "Batu", "Kai", "Sven", "Milan", "Goran", "Chris", "Nico",
"Dennis", "Emin", "Luca"]

Implementieren Sie eine Funktion entferne_spieler(), mit der ein Spieler aus dem Array entfernt wird.

Verwenden Sie für die Implementierung Ihrer Lösung die Datei L1_3_6_vorlage_volleyball_spieler_entfernen.py, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen L1_3_6_volleyball_spieler_entfernen.py.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

Array ohne den entfernten Eintrag

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Index der Stelle, an welcher der Eintrag entfernt werden soll

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten? (Variablenliste)

Bedeutung	**Typ/Struktur**	**Variable/Größe**
Anzahl der Elemente des Arrays *kader*	Ganzzahl	laenge
Index des zu entfernenden Eintrags	Ganzzahl	index

(4) Gewünschter Ablauf des Programms mit Beispieldaten:

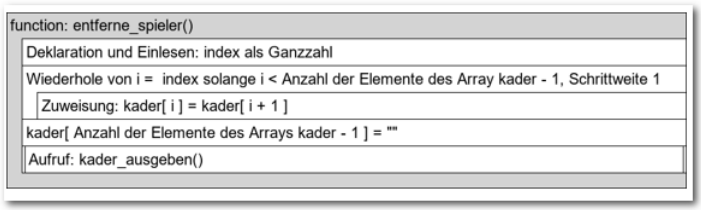
Funktion entferne_spieler()	
Eingabe	Entferne Spieler an der Stelle mit dem Index: 4
Ausgabe	---------- Kader ---------- Armin Batu Kai Sven Milan Goran Chris Nico Dennis Emin Luca

(5) Verarbeitung

- Arrayelement, das gelöscht werden soll (index), wird von dem Arrayelement dahinter (index + 1) überschrieben.
- Alle folgenden Arrayelemente werden wiederum von dem darauffolgenden Arrayelement überschrieben.
- Das letzte Element im Array kader erhält einen „leeren" Eintrag.

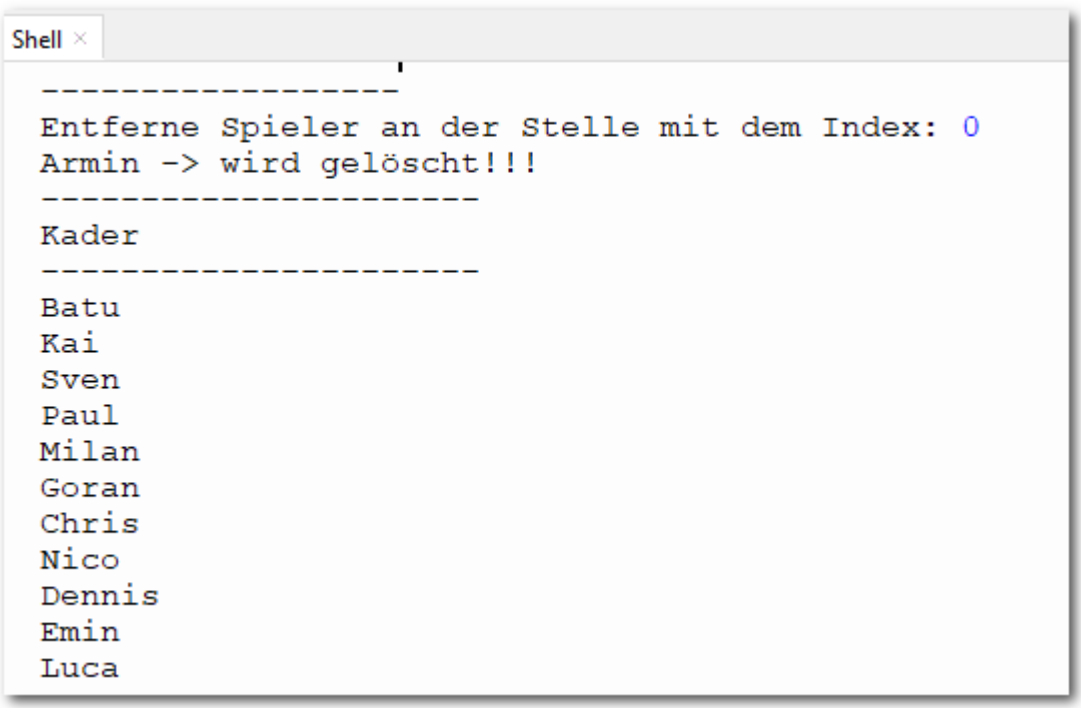
(III) Struktogramm

Das Array kader und die Funktion zeige_kader() sind bereits implementiert!



(IV) Programmcode (Python-Code)

```
107 def entferne_spieler():
108     print("------------------")
109     #EINGABEN
110     index = int(input("Entferne Spieler an der Stelle mit dem Index: ?"))
111     #Rücke den Spieler dann Schritt für Schritt nach vorn
112     for i in range(index, len(kader)-1):
113         kader[i] = kader[i+1]
114 
115     #letztes Element mit nichts überschreiben
116     kader[len(kader)-1] = ""
117     zeige_kader()
118 
```



4 Grundlagen Algorithmik L2 1

Grundlagen Algorithmik L2 1

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L2 1 Arbeitsauftrag Einführung Algorithmik

L2_1 Einführung Algorithmik

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Aufgabenstellungen das Informations-material L2_1 Information_Algorithmik.docx.

1 Formulieren Sie einen Satz, der den Begriff 'Algorithmus' definiert.

Ein Algorithmus ist eine Handlungsanweisung, die eindeutig den Weg zur Lösung eines gegebenen Problems beschreibt.

2 Erstellen Sie eine Übersicht der Eigenschaften von Algorithmen.

- *Allgemeinheit*
  Ein Algorithmus muss eine Vielzahl von Problemen der gleichen Art lösen.
- *Eindeutigkeit*
  Ein Algorithmus muss eindeutig sein. Er darf keine widersprüchliche Beschreibung haben.
- *Ausführbarkeit*
  Jeder Einzelschritt eines Algorithmus muss ausführbar sein.
- *Endlichkeit*
  Die Beschreibung eines Algorithmus muss endlich sein.
- *Terminierung*
  Ein Algorithmus muss nach endlich vielen Schritten enden und ein Ergebnis liefern.
- *Determiniertheit*
  Ein Algorithmus muss bei gleichen Voraussetzungen immer das gleiche Ergebnis liefern.
- *Determinismus*
  Für jeden Schritt eines Algorithmus darf höchstens eine Möglichkeit der Fortsetzung bestehen.

3.1 Der Möbelhersteller Holzwurm GmbH möchte für den Mitnahmetisch ALPHA eine Aufbauanleitung erstellen.

Formulieren Sie anhand der 'Abbildungen zum Aufbau des Mitnahmetisches ALPHA' (siehe Anlage, Seite 2) einen Algorithmus, der die Vorgehensweise zum Aufbau des Tisches beschreibt.

Aufbauanleitung:

- Überprüfung der mitgelieferten Teile auf Vollständigkeit.
- Filzgleiter in die Öffnung an der Unterseite des Tischbeines stecken.
- Wiederholung des Vorgangs für die anderen drei Tischbeine.
- Tischplatte mit der Unterseite nach oben auf den Boden legen.
- Tischbein mit der Oberseite auf die vorgebohrten Löcher der Tischplatte setzen.
- Vier Befestigungsschrauben an den vorgesehenen Löchern ansetzten und fest anziehen.
- Wiederholung des Vorgangs für die anderen drei Tischbeine.
- Tisch umdrehen und auf die Beine stellen.

3.2 Die Kfz-Steuer für PKWs sollen mit Hilfe eines Programms ermittelt und ausgege ben werden.

Für die Berechnung der Steuer gelten folgende (vereinfachte) Regeln:

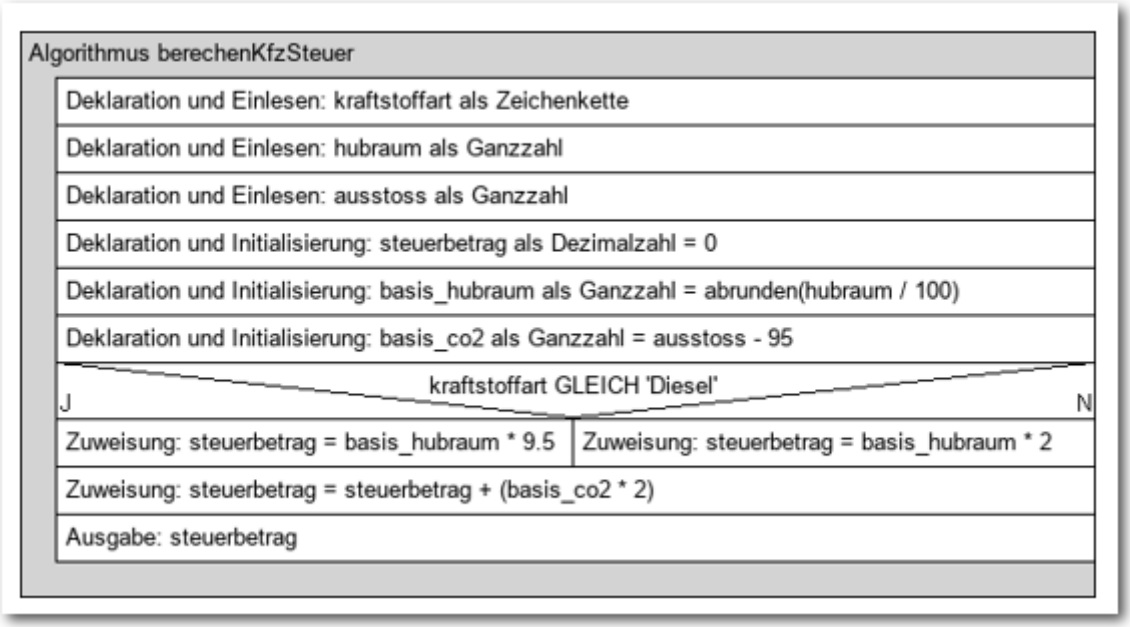
Für Dieselfahrzeuge werden pro 100 Kubikzentimeter Hubraum 9,50 EUR, für Benzinfahr-zeuge 2,00 EUR fällig.

Zusätzlich wird der CO2-Ausstoß des Fahrzeugs bei der Steuerberechnung berücksichtigt. Pro Gramm CO2/km werden 2,00 EUR berechnet. Dabei ist ein CO2-Freibetrag von 95 g/km zu beachten.

Für ein Diesel-Fahrzeug mit 2090 ccm und einem CO2-Ausstoß von 174 g/km ergibt sich somit ein Steuerbetrag von 348,00 EUR [20 * 9,50 + (174 – 95) * 2]

Formulieren Sie für die Berechnung der Kfz-Steuer einen Algorithmus in Form eines Struktogramms.

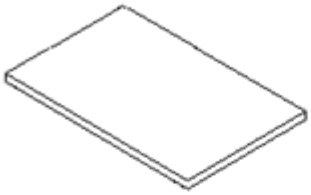
Lösung:


Algorithmus berechenKfzSteuer
Deklaration und Einlesen: kraftstoffart als Zeichenkette
Deklaration und Einlesen: hubraum als Ganzzahl
Deklaration und Einlesen: ausstoss als Ganzzahl
Deklaration und Initialisierung: steuerbetrag als Dezimalzahl = 0
Deklaration und Initialisierung: basis_hubraum als Ganzzahl = abrunden(hubraum / 100)
Deklaration und Initialisierung: basis_co2 als Ganzzahl = ausstoss - 95
kraftstoffart GLEICH 'Diesel'
J
N
Zuweisung: steuerbetrag = basis_hubraum * 9.5
Zuweisung: steuerbetrag = basis_hubraum * 2
Zuweisung: steuerbetrag = steuerbetrag + (basis_co2 * 2)
Ausgabe: steuerbetrag

Anlage: Abbildungen zum Aufbau des Mitnahmetisches ALPHA

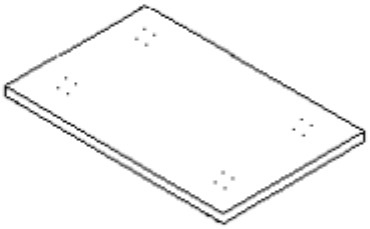
① Mitgelieferte Teile:

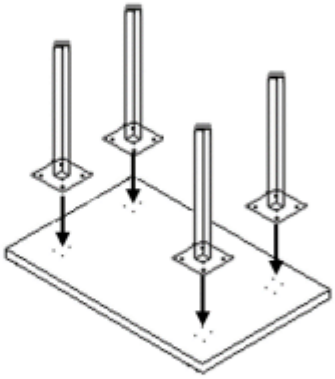
Tischplatte Ahornfurnier (1 Stück)	Tischbein Edelstahl (4 Stück)	Filzgleiter (4 Stück)	Befestigungsschrauben (16 Stück)

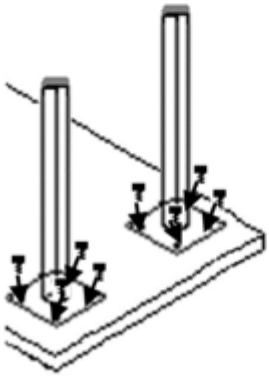


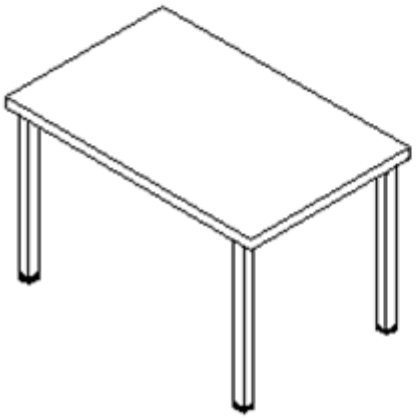

Aufbau:

② 

③ 

④ 

⑤ 

⑥ 

5 Sortieralgorithmen L2 2

Sortieralgorithmen L2 2

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L2 2 1 1 Arbeitsauftrag Bubble Sort Lotto

L2_2.1.1 Sortierung: Bubble Sort – Lottozahlen

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Aufgaben¬stellungen die Informationsmaterialien

- L2_2.1 Information_Bubble_Sort.docx
- L2_2.1 Präsentation_Prinzip_Bubble_Sort.ppsx.

(I) Problemstellung

Die Zahlen der aktuellen Lottoziehung liegen in der Reihenfolge der Ziehung in einem Array lotto vor (48, 5, 17, 32, 7, 29) und sollen noch sortiert werden. Sie erhalten den Auftrag, ein entsprechendes Programm zu entwickeln, das die Lottozahlen mit Hilfe des Sortieralgorithmus 'Bubble Sort' sortiert und die sortierten Zahlen in der Konsole ausgibt

Verwenden Sie für die Implementierung Ihrer Lösung die Datei L2_2_1_1_vorlage_bubble_sort_lotto.py, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen L2_2_1_1_bubble_sort_lotto.py.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

Sortiertes Array lotto[ ]

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Keine

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten?
(Variablenliste)

Bedeutung	Typ/Struktur	Variable/Größe
Lottozahlen	Array	lotto
Anzahl Elemente des Arrays lotto	Ganzzahl	laenge
Zwischenspeicher	Ganzzahl	zwischenspeicher

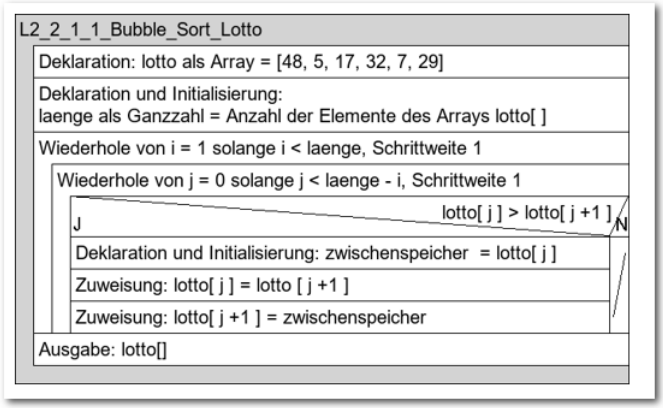
(4) Gewünschter Ablauf des Programms mit Beispieldaten:

Ausgabe	`[5, 7, 17, 29, 32, 48]`

(5) Verarbeitung

Sortiere mit dem Bubble Sort das Array lotto.

(III) Struktogramm



(IV) Programmcode (Python-Code)

EIGENE LÖSUNG:

Musterlösung 1:

```
lotto = [48, 5, 17, 32 , 7, 29]
laenge = len(lotto)

#Sortieren
for i in range(1,laenge):
  for j in range(0,laenge-i):
    if lotto[j] > lotto[j+1]:
      zwischenspeicher = lotto[j]
```

```
        lotto[j] = lotto[j+1]
        lotto[j+1] = zwischenspeicher

#Ausgabe
print(lotto)
```

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L2 2 1 2 Arbeitsauftrag Bubble Sort Zahlenreihe

L2_2.1.2 Sortierung: Bubble Sort – Zahlenreihe

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Aufgabenstellungen die Informationsmaterialien

L2_2.1 Information_Bubble_Sort.docx

L2_2.1 Präsentation_Prinzip_Bubble_Sort.ppsx.

(I) Problemstellung

Sie sollen ein Programm schreiben, das fünf Zahlen einliest und diese wieder sortiert ausgibt. Implementieren Sie für die Sortierung den Bubble Sort.

Verwenden Sie für die Implementierung Ihrer Lösung die Datei L2_2_1_2_vorlage_bubble_sort_zahlenreihe.py, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen

L2_2_1_2_bubble_sort_zahlenreihe.py.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

Sortiertes Array

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Fünf Zahlen

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten?

(Variablenliste)

Bedeutung	Typ/Struktur	Variable/Größe
Zahlen	Array	zahlen
Anzahl der Elemente des Arrays zahlen	Ganzzahl	laenge
Zwischenspeicher	Ganzzahl	zwischenspeicher

(4) Gewünschter Ablauf des Programms mit Beispieldaten:

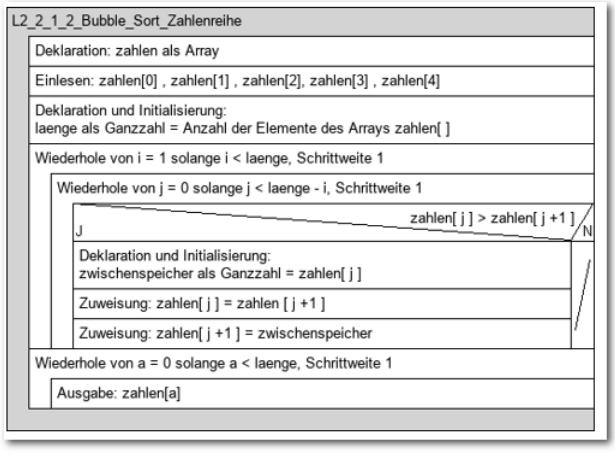
Eingabe	Zahl 1: 5 Zahl 2: 6 Zahl 3: 1 Zahl 4: 2 Zahl 5: 3 Zahl 6: 4

Ausgabe	1 2 3 4 5 6

(5) Verarbeitung

Sortiere mit dem Bubble Sort das Array zahlen.

(III) Struktogramm



(IV) Programmcode (Py thon-Code)

EIGENE LÖSUNG!

Python_II_L2_2_1_2_BubbleSort_Zahlenreihe.py

```
#EINGABE: Ein leeres Array deklarieren
zahlen = []
#EINGABE: Initialilisierung des Arrays
zahlen.append(int(input("Zahl 1: ")))
zahlen.append(int(input("Zahl 2: ")))
zahlen.append(int(input("Zahl 3: ")))
zahlen.append(int(input("Zahl 4: ")))
zahlen.append(int(input("Zahl 5: ")))
zahlen.append(int(input("Zahl 6: ")))

#VERARBEITUNG: Funktion
def bubblesort():
    print("----------------")
    laenge = len(zahlen)

    #nimmt sich zwei nebenliegende Elemente von vorne beginn
    #und tauscht ggf
    #wiederholt bis nicht mehr getauscht werden muss
    for i in range(1, laenge,+1):
        for j in range(0, laenge-i,+1):
            #Tauschen
            if zahlen[j] > zahlen[j+1]:
                zwischenspeicher = zahlen[j]
                zahlen[j] = zahlen[j+1]
                zahlen[j+1] = zwischenspeicher
    for a in range(0,laenge,+1):
        print(zahlen[a])

#VERARBEITUNG: Zusatz - Funktion/Methode für die senkrechte
def zeige_zahlen():
    print("------------------------")
    print("Zahlen")
    print("------------------------")
    i = 0
    #for i in range(len(lotto)):
    #print(lotto[i])
    while i < len(zahlen):
        print(zahlen[i])
        i=i+1
#####AUSGABE
#L2 2 1 2 Methodenaufruf bubblesort
bubblesort()
```

```
Python 3.7.5 (bundled)
>>> %Run Python_II_L2_2_1_

  Zahl 1: 5
  Zahl 2: 6
  Zahl 3: 1
  Zahl 4: 2
  Zahl 5: 3
  Zahl 6: 4
  ----------------
  1
  2
  3
  4
  5
  6
```

Musterlösung 1:

```
     if zahlen[j] > zahlen[j+1]:
        zwischenspeicher = zahlen[j]
        zahlen[j] = zahlen[j+1]
        zahlen[j+1] = zwischenspeicher

#Ausgabe
for i in range(laenge):
   print(zahlen[i])
# Array deklarieren
zahlen = []

#Eingabe
zahlen.append(int(input("Zahl 1: ")))
zahlen.append(int(input("Zahl 2: ")))
zahlen.append(int(input("Zahl 3: ")))
zahlen.append(int(input("Zahl 4: ")))
zahlen.append(int(input("Zahl 5: ")))

laenge = len(zahlen)

#Sortierung
for i in range(1, laenge):
   for j in range(laenge-i):
```

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L2 2 2 2 Arbeitsauftrag Selection Sort Zahlenreihe

L2_2.2.2 Sortierung: Selection Sort – Zahlenreihe

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Aufgabenstellungen die Informationsmaterialien L2_2.2 Information_Selection_Sort.docx

(I) Problemstellung

Sie sollen ein Programm schreiben, das fünf Zahlen einliest und diese wieder sortiert ausgibt. Implementieren Sie für die Sortierung den Selection Sort.

Verwenden Sie für die Implementierung Ihrer Lösung die Datei L2_2_2_2_vorlage_selection_sort_zahlenreihe.py, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen

L2_2_2_2_selection_sort_zahlenreihe.py.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

Sortiertes Array

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Fünf Zahlen

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten?
(Variablenliste)

Bedeutung	Typ/Struktur	Variable/Größe
Zahlen	Array	zahlen
Anzahl der Elemente des Arrays zahlen	Ganzzahl	laenge
Aktuelle Zahl (bzw. deren Index)	Ganzzahl	akt_idx
kleinste Zahl (bzw. deren Index)	Ganzzahl	min_idx
Zwischenspeicher	Ganzzahl	zwischenspeicher

(4) Gewünschter Ablauf des Programms mit Beispieldaten:

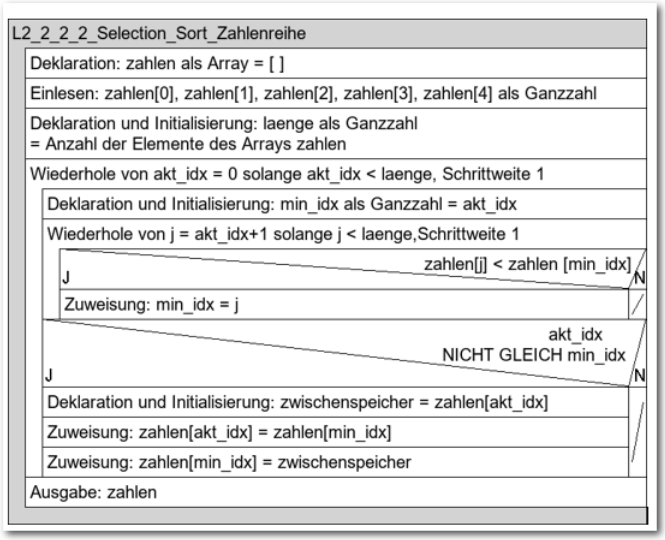
Eingabe	Zahl 1: 5 Zahl 2: 6 Zahl 3: 1 Zahl 4: 2 Zahl 5: 3

Ausgabe	[1, 2, 3, 5, 6]

(5) Verarbeitung

Sortiere mit dem Selection Sort das Array zahlen.

(III) Struktogramm



(IV) Programmcode (Python-Code)

EIGENE LÖSUNG!

Python_II_L2_2_2_2_SelectionSort_Zahlenreihe.py

```
zahlen = []
#EINGABE: Initialilisierung des Arrays
zahlen.append(int(input("Zahl 1: ")))
zahlen.append(int(input("Zahl 2: ")))
zahlen.append(int(input("Zahl 3: ")))
zahlen.append(int(input("Zahl 4: ")))
zahlen.append(int(input("Zahl 5: ")))
zahlen.append(int(input("Zahl 6: ")))

#VERARBEITUNG: Funktion
def selectionsort():
    print("----------------")
    laenge = len(zahlen)

    #beginnt vorne
    #sucht min in restkette
    #tauscht
    #wiederholt bis nicht mehr getauscht werden muss
    for akt_idx in range(0, laenge,+1):
        min_idx = akt_idx #min setzen
        #sucht in der restkette
        for j in range(akt_idx+1,laenge,+1):
            if(zahlen[j] < zahlen[min_idx]):
              min_idx = j #min ggf neu setzen
        if(akt_idx != min_idx):#und tauscht ggf.
            zwischenspeicher = zahlen[akt_idx]
            zahlen[akt_idx] = zahlen[min_idx]
            zahlen[min_idx] = zwischenspeicher
    print(zahlen)

#VERARBEITUNG: Zusatz - Funktion/Methode für
#die senkrechte Ausgabe
def zeige_zahlen():
    print("------------------------")
    print("Zahlen")
    print("------------------------")
    i = 0
    #for i in range(len(lotto)):
    #print(lotto[i])
    while i < len(zahlen):
        print(zahlen[i])
        i=i+1
#####AUSGABE
#L2 2 1 2 Methodenaufruf selectic
selectionsort()
```

Shell

```
>>> %Run Python_II_L2_2_2_2_SelectionSort_Zahlenreihe.py

  Zahl 1: 2
  Zahl 2: 4
  Zahl 3: 6
  Zahl 4: 3
  Zahl 5: 4
  Zahl 6: 7
  ----------------
  [2, 3, 4, 4, 6, 7]

>>>
```

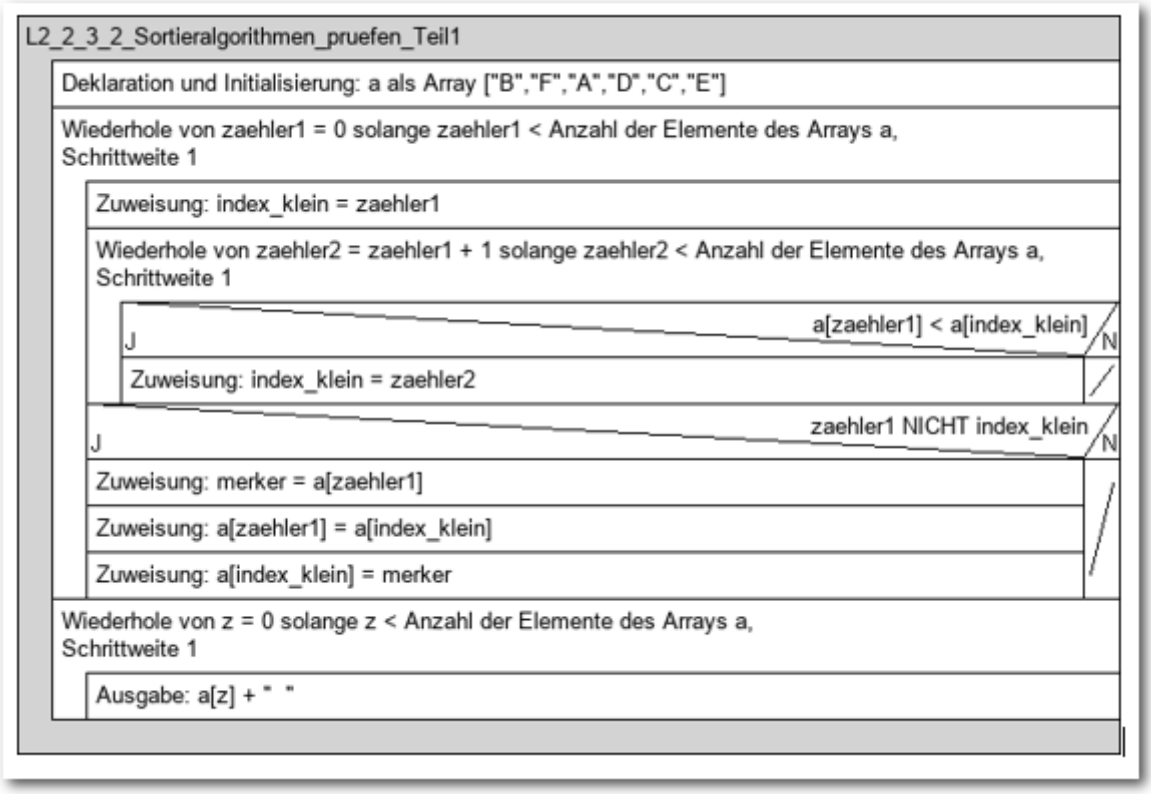
Musterlösung 1:

```
# Array deklarieren und initialisieren
zahlen = []
# Zahlen einlesen
zahlen.append(int(input("Zahl 1: ")))
zahlen.append(int(input("Zahl 2: ")))
zahlen.append(int(input("Zahl 3: ")))
zahlen.append(int(input("Zahl 4: ")))
zahlen.append(int(input("Zahl 5: ")))

#Sortierung
laenge = len(zahlen)

for akt_idx in range(laenge):
   min_idx = akt_idx
   for j in range(akt_idx + 1,laenge):
      #Suche nach der niedrigsten Zahl
      if zahlen[j] < zahlen[min_idx]:
         min_idx = j
   #Tausch
   if akt_idx != min_idx:
      zwischenspeicher = zahlen[akt_idx]
      zahlen[akt_idx] = zahlen [min_idx]
      zahlen[min_idx] = zwischenspeicher

#Ausgabe
print(zahlen)
```

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L2 2 3 2 Arbeitsauftrag Sortieralgorithmen prüfen

L2_2.3.2 Sortieralgorithmen prüfen

1 Ihnen liegt nachfolgendes Struktogramm zur Analyse vor:



1.1. Analysieren Sie den im Struktogramm dokumentierten Programmablauf und beschreiben Sie das Ergebnis des Algorithmus.

Der Algorithmus sortiert das angegebene Array in alphabetischer Reihenfolge.

1.2. Implementieren Sie den Programmcode gemäß des abgebildeten Struktogramms.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen L2_2_3_2_sortieralgorithmen_pruefen_teil1.py.

EIGENE LÖSUNG!

L2_2_3_3_Sortieralgorithmen_pruefen_teil2.py | L2_2_3_3_Sortieralgorithmen_pruefen_teil1.py

```
1  #Deklaration und Initialisierung des Arrays
2  a = ["B","F","A","D","C","E"]
3
4  #Methode/Funktion für die Sortierung mit dem Selectsort
5  def sortieralgorithmen_pruefen_teil1():
6      for zaehler1 in range(0,len(a),+1):
7          index_klein = zaehler1
8          for zaehler2 in range((zaehler1+1),len(a),+1):
9              #Korrektur: zaehler2 NICHT zaehler1
10             if(a[zaehler2] < a[index_klein]):
11                 index_klein = zaehler2
12         if(zaehler1 != index_klein):
13             merker = a[zaehler1]
14             a[zaehler1] = a[index_klein]
15             a[index_klein] = merker
16     #AUSGABE
17     for z in range(0,len(a),+1):
18         print(a[z] + " ")
19
20 #Funktions-/Methodenaufruf
21 sortieralgorithmen_pruefen_teil1()
```

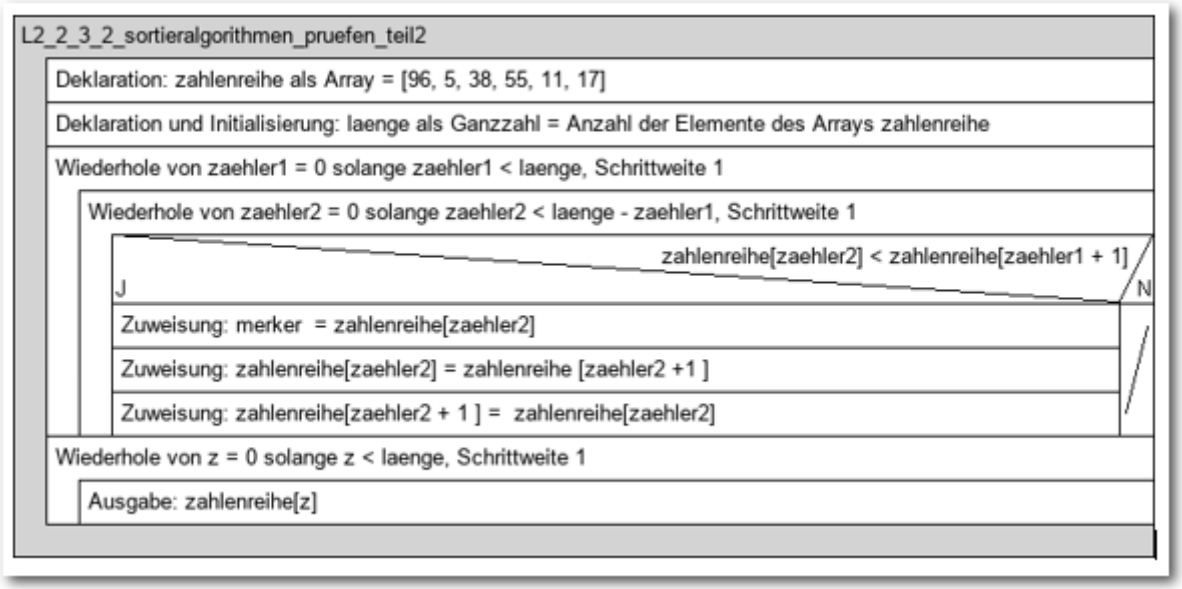
Shell

```
>>> %Run L2_2_3_3_Sortieralgorithmen_pruefen_teil1.py
A
B
C
D
E
F

>>>
```

Musterlösung Moodle-Kurs:

```
a = ["B", "F", "A", "D", "C", "E"]

    for zaehler1 in range(len(a)):
      index_klein = zaehler1

      for zaehler2 in range(zaehler1+1, len(a)):
        if a[zaehler2] < a[index_klein]:
          index_klein = zaehler2
      if zaehler1 != index_klein:
        merker = a[zaehler1]
        a[zaehler1] = a[index_klein]
        a[index_klein] = merker

    for z in range(len(a)):
      print(a[z])
```

2 Ihnen liegt folgendes Struktogramm zur Analyse vor:



2.1 Nennen Sie die Zielsetzung des dargestellten Algorithmus.

Absteigende Sortierung mit Bubble Sort.

2.2 Analysieren Sie die einzelnen Anweisungen des dargestellten Algorithmus und lokalisieren Sie vorhandene logische Fehler.

EIGENE LÖSUNG (gemeinsame Schülerlösung 1)!

1. Schleifenkopf:

Keine Korrektur.

2.Schleifenkopf:

Korrektur:

→ for zaehler2 in range(0,laenge-1,+1):

Bedingung IF ELSE:

Korrektur der Bedingung:

→ if(zahlenreihe[zaehler1] > zahlenreihe[zaehler2]):

Tauschalgorithmus:

Korrektur:

→ merker = zahlenreihe[zaehler2]

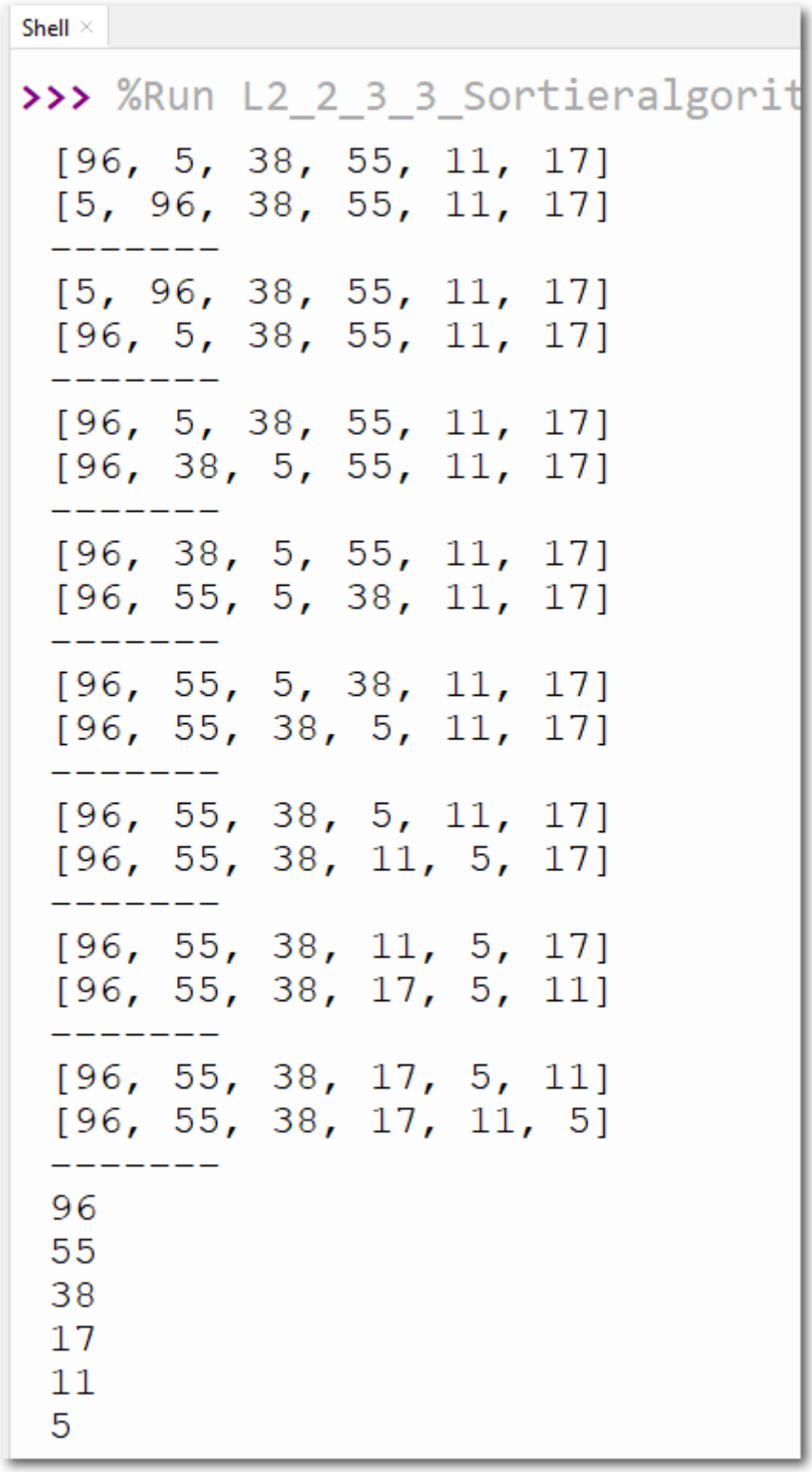
→ zahlenreihe[zaehler2] = zahlenreihe[zaehler1]

→ zahlenreihe[zaehler1] = merker

EIGENE LÖSUNG - Quellcode(gemeinsame Schülerlösung 1)!

L2_2_3_3_Sortieralgorithmen_pruefen_teil2.py

```
#Deklaration und Initialisierung des Arrays
zahlenreihe = [96,5,38,55,11,17]

#Methode/Funktion für die Sortierung mit dem Selectsort
def sortieralgorithmen_pruefen_teil2():
    laenge = len(zahlenreihe)
    for zaehler1 in range(0,laenge,+1):
        for zaehler2 in range(0,laenge-1,+1):
            #hat hier zaehler1 schon um 1 inkrementiert!
            #rechtes Element > linkes Element
            if(zahlenreihe[zaehler1] > zahlenreihe[zaehler2]):
                #Testausgabe: vor dem Tausch
                print(zahlenreihe)
                #Tauschen
                merker = zahlenreihe[zaehler2]
                zahlenreihe[zaehler2] = zahlenreihe[zaehler1]
                zahlenreihe[zaehler1] = merker
                #Testausgabe: nach dem Tausch
                print(zahlenreihe)
                print("-------")
    #AUSGABE
    for z in range(0,laenge,+1):
        print(zahlenreihe[z])

#Funktions-/Methodenaufruf
sortieralgorithmen_pruefen_teil2()
```


Shell
>>> %Run L2_2_3_3_Sortieralgorit
[96, 5, 38, 55, 11, 17]
[5, 96, 38, 55, 11, 17]

[5, 96, 38, 55, 11, 17]
[96, 5, 38, 55, 11, 17]

[96, 5, 38, 55, 11, 17]
[96, 38, 5, 55, 11, 17]

[96, 38, 5, 55, 11, 17]
[96, 55, 5, 38, 11, 17]

[96, 55, 5, 38, 11, 17]
[96, 55, 38, 5, 11, 17]

[96, 55, 38, 5, 11, 17]
[96, 55, 38, 11, 5, 17]

[96, 55, 38, 11, 5, 17]
[96, 55, 38, 17, 5, 11]

[96, 55, 38, 17, 5, 11]
[96, 55, 38, 17, 11, 5]

96
55
38
17
11
5

EIGENE LÖSUNG (gemeinsame Schülerlösung 2)!

2.Schleifenkopf:

Korrektur → for zaehler2 in range(zaehler1+1,laenge,+1):

Bedingung IF ELSE:

Korrektur → zahlenreihe[zaehler2] > zahlenreihe[zaehler1]

Tauschalgorithmus:

Korrektur → merker = zahlenreihe[zaehler2]

Korrektur → zahlenreihe[zaehler2] = zahlenreihe[zaehler1]

Korrektur → zahlenreihe[zaehler1]= merker

EIGENE LÖSUNG - Quellcode(gemeinsame Schülerlösung 2)!

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L2 2 3 3 Arbeitsauftrag Volleyballspieler sortieren

L2_2.3.3 Volleyball – Spielernamen sortieren

(I) Problemstellung

Die Software des Trainers der Abteilung Volleyball des Sportvereines Mühlberger SC enthält ein Array mit allen Spielernamen des Mannschaftskaders.

kader = ["Nico", "Batu", "Paul", "Kai", "Sven", "Milan", "Goran", "Chris", "Armin", "Dennis", "Emin", "Luca"]

Mit Hilfe der Software soll es möglich sein, die Spieler in alphabetischer Reihenfolge auszugeben.

Implementieren Sie ein Programm, mit dem das Array kader in alphabetischer Reihenfolge sortiert und in der Konsole ausgegeben wird.

Verwenden Sie für die Implementierung Ihrer Lösung die Datei L2_2_3_3_vorlage_volleyball_spieler _sortieren.py, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen L2_2_3_3_volleyball_spieler_sortieren.py.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

Sortiertes Array

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Keine

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten?
(Variablenliste)

Bedeutung	Typ/Struktur	Variable/Größe
Index für aktuelle Stelle im Array	Ganzzahl	i
Index für Stelle des „kleinsten" Namens	Ganzzahl	minindex
Zwischenspeicher	Zeichenanzahl	zwischenspeicher

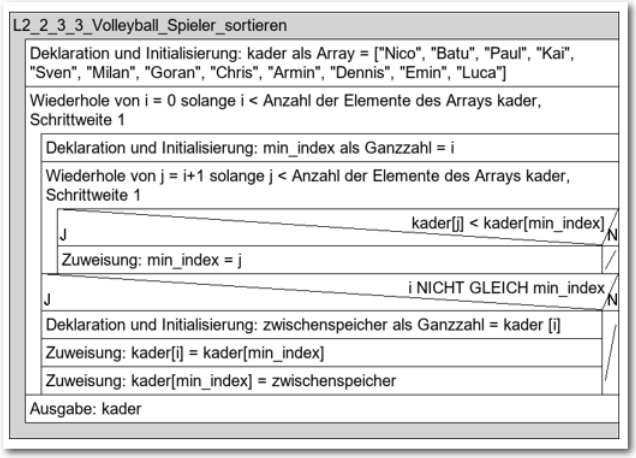
(4) Gewünschter Ablauf des Programms mit Beispieldaten:

Ausgabe	Kader alphabetisch sortiert: ['Armin', 'Batu', 'Chris', 'Dennis', 'Emin', 'Goran', 'Kai', 'Luca', 'Milan', 'Nico', 'Paul', 'Sven']

(5) Verarbeitung

- Suche das kleinste Element im Array kader und speichere den Index (min_index).
- Tausche das kleinste Element (min_index), mit dem ersten Element im Array.
- Suche das zweitkleinste Element im Array kader und speichere den Index (min_index).
- Tausche das zweitkleinste Element (min_index), mit dem zweiten Element im Array.
- usw…

(III) Struktogramm



(IV) Programmcode (Python-Code)

L2_2_1_1_Bubble_Sort_Lottozahlen.py | Python_II_L2_2_3_3_Sortieralgorithmen_Volleyballspielernamen_sortieren.py

```
121 def sortiere_spieler():#selection sort
122     for i in range(0, len(kader),+1):
123         #setze Minimum
124         min_index = i
125         #suche in der Restliste
126         for j in range(i+1, len(kader),+1):
127             # setzt minimum ggf neu
128             if(kader[j] < kader[min_index]):
129                 min_index = j
130         if(i != min_index):
131             zwischenspeicher = kader[i]
132             kader[i] = kader[min_index]
133             kader[min_index] = zwischenspeicher
134     print(kader)
135
```

Gemeinsame Schülerlösung mit dem Bubblesort:

L2_2_3_3_3_Sortieralgorithmen_Volleeyballspieler_sortieren.py

```
1  #Deklaration und Initialisierung des Arrays
2  kader = ["Nico", "Batu", "Paul", "Kai", "Sven", "Milan", "Goran", "Chris",
3           "Armin", "Dennis", "Emin", "Luca"]
4
5
6  #Methode/Funktion für die Sortierung mit dem Bubblesort
7  def bubblesort_kader():
8      laenge = len(kader)
9      for i in range(0,laenge,+1):
10         for j in range(0,laenge-1,+1):
11             #hat hier i schon um 1 inkrementiert!
12             #rechtes Element > linkes Element
13             if(kader[i] < kader[j]):
14                 merker = kader[j]
15                 kader[j] = kader[i]
16                 kader[i] = merker
17
18     #AUSGABE
19     print(kader)
20
21 #Funktions-/Methodenaufruf
22 bubblesort_kader()
```

Shell

```
>>> %cd 'D:\Schule\Unterricht\Wirtschaftsinformatik\2020_INF_AG01_AG13\Arbeitsmate
      rialien\Meine_Loesungen'
>>> %Run L2_2_3_3_3_Sortieralgorithmen_Volleeyballspieler_sortieren.py
 ['Armin', 'Batu', 'Chris', 'Dennis', 'Emin', 'Goran', 'Kai', 'Luca', 'Milan', 'Nico', '
 Paul', 'Sven']
```

Musterlösung 1:

```
kader = ["Nico", "Batu", "Paul", "Kai", "Sven", "Milan", "Goran", "Chris",  "Ar-
min", "Dennis", "Emin", "Luca"]


for i in range(len(kader)):
  min_index = i
  for j in range(i+1,len(kader)):
    if kader[j] < kader[min_index]:
      min_index = j
```

```
    if i != min_index:
        zwischenspeicher = kader[i]
        kader[i] = kader[min_index]
        kader[min_index] = zwischenspeicher

#Ausgabe
print("Kader alphabetisch sortiert: ")
print(kader)
```

6 Suchalgorithmen L2 3

Suchalgorithmen L2 3

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L2 3 1 1 Arbeitsauftrag Lineare Suche Mitgliedsnummer

L2_3.1.1 Suchen: Lineare Suche – Mitgliedsnummer

Der Sportverein Mühlberger SC hat seine Mitglieder zur Mitgliederversammlung eingeladen. Zugang zur Veranstaltung sollen nur Mitglieder haben. Deswegen wird am Eingang der Mitgliederausweis kontrolliert.

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Aufgabenstellungen die Informationsmaterialien

- L2_3.1 Information_Lineare_Suche.docx
- L2_3.1 Präsentation_Prinzip_Lineare_Suche.ppsx.

(I) Problemstellung

Implementieren Sie ein Programm, das überprüft, ob ein Besucher der Mitgliederversammlung zu-gangsberechtigt ist. Dafür soll am Eingang die Mitgliedsnummer in das Programm eingegeben werden. Nach der Eingabe der Mitgliedsnummer wird geprüft, ob die eingegebene Nummer existiert. Alle ver¬gebenen Mitgliedsnummern des Vereins sind im Array mnr erfasst.

mnr = [1001, 1019, 1014, 1009, 1005, 1002, 1018, 1008, 1003, 1010, 1007, 1004, 1020, 1013, 1015, 1011, 1017, 1012, 1006, 1016]

Wird die eingegebene Mitgliedsnummer gefunden, soll die Meldung „Zutritt gewährt" ausgegeben werden. Wird die Nummer nicht gefunden, soll die Meldung „Zutritt verweigert" erscheinen.

Verwenden Sie für die Implementierung Ihrer Lösung die Datei L2_3_1_1_vorlage_lineare_suche _mitgliedsnummer.py, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen L2_3_1_1_loesung_lineare_suche_mitgliedsnummer.py.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

"Zutritt gewährt" bzw. " Zutritt verweigert"

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Mitgliedsnummer

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten?
(Variablenliste)

Bedeutung	**Typ/Struktur**	**Variable/Größe**
Vergebene Mitgliedsnummern	Array	mnr
Gesuchte Mitgliedsnummer	Ganzzahl	nummer
Anzahl der Elemente des Arrays mnr	Ganzzahl	laenge
Merker für Suchergebnis	Boolean	gefunden

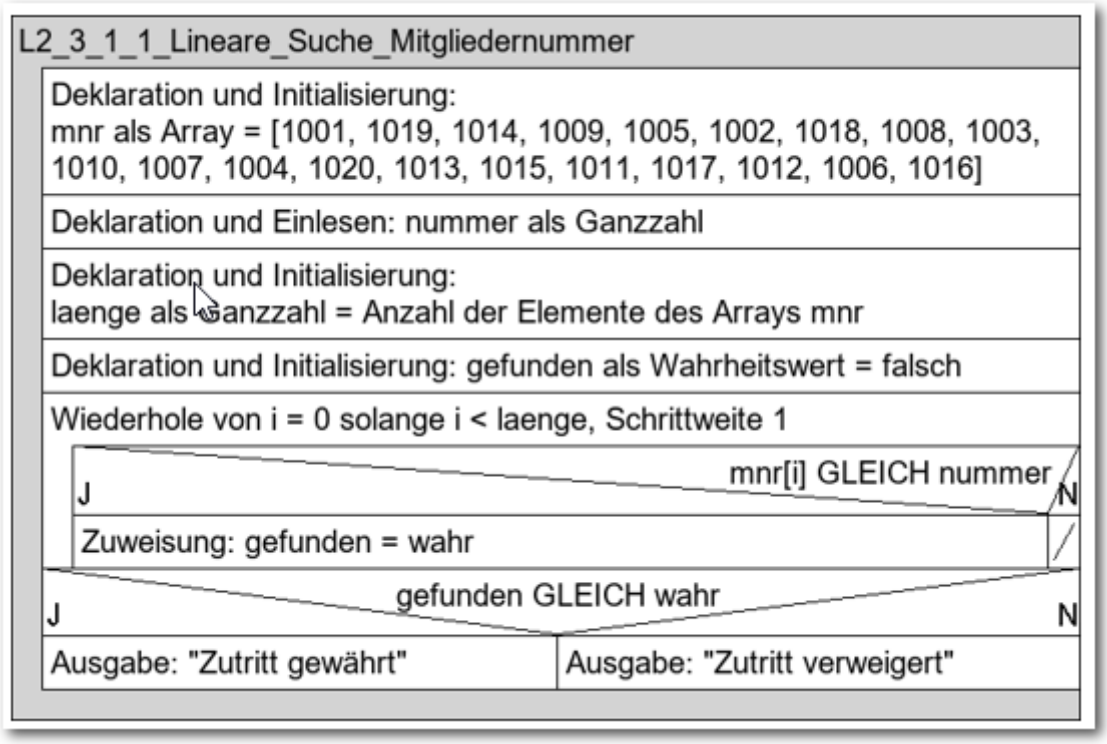
(4) Gewünschter Ablauf des Programms mit Beispieldaten:

Mitgliedsnummer wird gefunden		Mitgliedsnummer wird nicht gefunden	
Eingabe	Mitgliedsnummer eingeben: 1001	**Eingabe**	Mitgliedsnummer eingeben: 20
Ausgabe	Zutritt gewährt	**Ausgabe**	Zutritt verweigert

(5) Verarbeitung

- Vergleiche jedes Arrayelement mit der gesuchten Nummer.
- Ist ein Arrayelement und die gesuchte Nummer identisch, ist die Nummer gefunden (gefunden = True).

(III) Struktogramm



(IV) Programmcode (Python-Code)

Python_II_L2_3_1_1_Lineare_Suche_Mitgliedsnummer.py

```
#Deklaration und Initialisierung der Eingabewerte
mnr = [1001,1019,1014,1009,1005,1002,1018,1008,1003,1010,1007,1004,1020,1013,1015,1011,1017,1012,1006,1016]
nummer = int(input("Mitgliedsnummer eingeben: "))
laenge = len(mnr)
gefunden = bool(False)

#Verarbeitung
def suche_linear(gefunden):
    for i in range(0,laenge,+1):
        if(mnr[i] == nummer):
            gefunden = True
    if(gefunden == True):
        print("Zutritt gewährt")
    else:
        print("Zutritt verweigert")

#Verarbeitung: Methodenaufruf
suche_linear(gefunden)
```

Shell

```
>>> %Run Python_II_L2_3_1_1_Lineare_Suche_Mitgliedsnummer.py
 Mitgliedsnummer eingeben: 1001
 Zutritt gewährt
>>> %Run Python_II_L2_3_1_1_Lineare_Suche_Mitgliedsnummer.py
 Mitgliedsnummer eingeben: 20
 Zutritt verweigert
```

#gemeinsame Schülerlösung:

L2_3_1_1_Suchalgorithmen_lineare_Suche_Migliedsnummer.py

```
1
2  #Verarbeitung
3  #Methode/Funktion deklariert und Implementiert
4  def lineare_suche():
5      #Deklaration und Initialisierung der Liste
6      a =  [1001, 1019, 1014, 1009, 1005, 1002, 1018, 1008, 1003, 1010, 100
7            1004, 1020, 1013, 1015, 1011, 1017, 1012, 1006, 1016]
8
9      #Deklaration und Initialisierung über die Eingabeaufforderung
10     gesuchte_zahl = int(input("Bitte geben Sie die Mitgliedsnummer ein:")
11
12     #Deklaration und Initialisierung eines Boolschen Wertes (Wahrheitswer
13     gefunden = False
14
15     #Deklaration und Initialisierung der Anzahl an Elementen im Array
16     laenge = len(a)
17
18     for i in range(0,laenge,+1):
19         if(a[i] == gesuchte_zahl):
20             gefunden = True
21     if(gefunden == True):
22         #Ausgabe
23         print("Zahl gefunden")
24     else:
25         print("Zahl nicht gefunden")
26
27 #Methodenaufruf
28 lineare_suche()
```

Shell

```
>>> %Run L2_3_1_1_Suchalgorithmen_lineare_Suche_Migliedsnummer.py
 Bitte geben Sie die Mitgliedsnummer ein:1002
 Zahl gefunden
>>> %Run L2_3_1_1_Suchalgorithmen_lineare_Suche_Migliedsnummer.py
 Bitte geben Sie die Mitgliedsnummer ein:23
 Zahl nicht gefunden
```

Moodle-Kurs-Musterlösung:

```
#Mitgliedsnummern
mnr = [1001, 1019, 1014, 1009, 1005, 1002, 1018, 1008, 1003, 1010, 1007,
       1004, 1020, 1013, 1015, 1011, 1017, 1012, 1006, 1016]

#Eingabe
nummer = int(input("Mitgliedsnummer eingeben: "))

laenge = len(mnr)
gefunden = False

#Suche
for i in range(laenge):
  if mnr[i] == nummer:
    gefunden = True

#Ausgabe
if gefunden:
  print("Zutritt gewährt")
else:
  print("Zutritt verweigert")
```

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L2 3 2 2 Arbeitsauftrag Binäre Suche Mitgliedsnummer

L2_3.2.2 Suchen: Binäre Suche – Mitgliedsnummer

Der Sportverein Mühlberger SC hat seine Mitglieder zur Mitgliederver¬sammlung eingeladen. Zugang zur Veranstaltung sollen nur Mitglieder haben. Deswegen wird am Eingang der Mitgliederausweis kontrolliert.

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Aufgabenstellungen die Informations-materialien

- L2_3.2 Information_Binäre_Suche.docx
- L2_3.2 Präsentation_Prinzip_Binäre_Suche.ppsx.

(I) Problemstellung

Mit der Aufgabenstellung aus "L2_3.1.1 Arbeitsauftrag Lineare Suche Mitgliedsnummer.docx" wurde bereits eine Lösung mit Hilfe der linearen Suche erarbeitet.

In dieser Aufgabenstellung soll nach der Eingabe einer Mitgliedsnummer mit Hilfe der binären Suche geprüft werden, ob die eingegebene Nummer existiert.

Alle vergebenen Mitgliedsnummern des Vereins sind im Array mnr erfasst.

mnr = [1001, 1019, 1014, 1009, 1005, 1002, 1018, 1008, 1003, 1010, 1007, 1004, 1020, 1013, 1015, 1011, 1017, 1012, 1006, 1016]

Wird die eingegebene Mitgliedsnummer gefunden, soll die Meldung „Zutritt gewährt“ ausgegeben werden. Wird die Nummer nicht gefunden, soll die Meldung „Zutritt verweigert“ erscheinen.

Verwenden Sie für die Implementierung Ihrer Lösung die Datei L2_3_2_2_vorlage_binaere_suche

_mitgliedsnummer.py, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen

L2_3_2_2_binaere_suche_mitgliedsnummer.py.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

"Zutritt gewährt" bzw. "Zutritt verweigert"

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Mitgliedsnummer

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten? (Variablenliste)

Bedeutung	Typ/Struktur	Variable/Größe
Vergebene Mitgliedsnummern (gegeben)	Array	mnr
Zwischenspeicher	Ganzzahl	zwischenspeicher
Gesuchte Mitgliedsnummer	Ganzzahl	nummer
Merker für Suchergebnis	Wahrheitswert	gefunden
Anzahl der Elemente des Arrays mnr	Ganzzahl	laenge
Startelement im Array mnr	Ganzzahl	index_anfang
Schlusselemet im Array mnr	Ganzzahl	index_ende
Mittleres Element im Array mnr	Ganzzahl	index_mitte

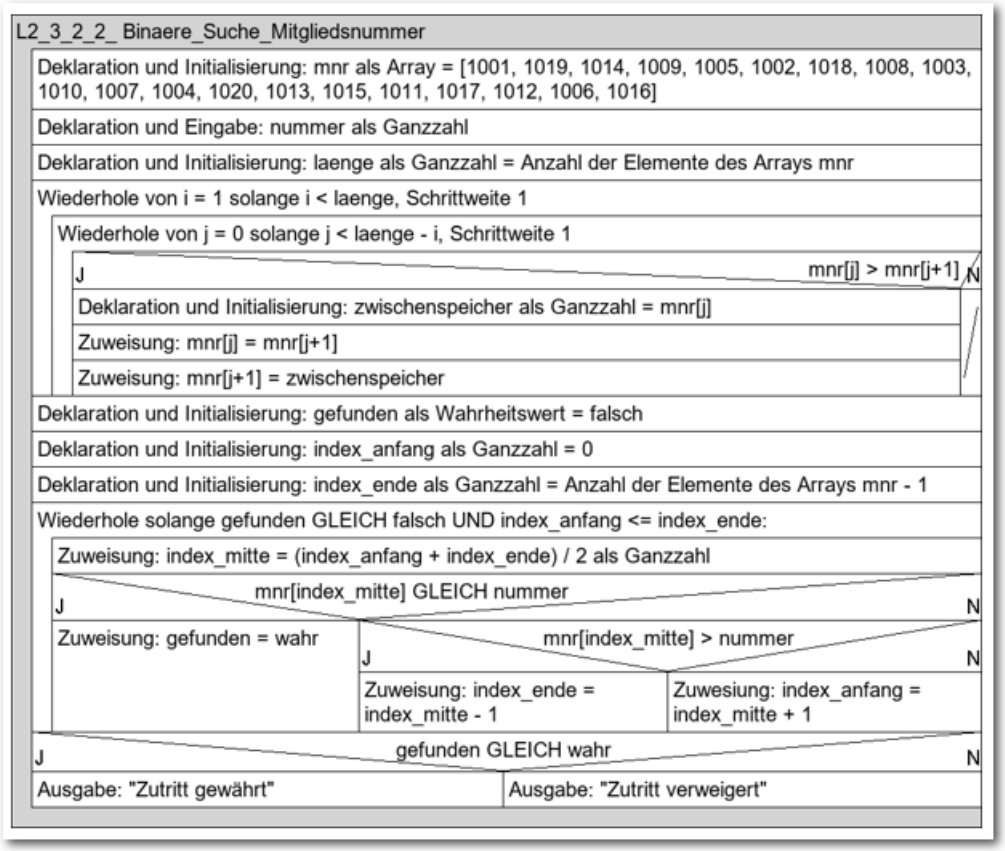
(4) Gewünschter Ablauf des Programms mit Beispieldaten:

Mitgliedsnummer wird gefunden		Mitgliedsnummer wird nicht gefunden	
Eingabe	Mitgliedsnummer eingeben: 1001	**Eingabe**	Mitgliedsnummer eingeben: 20
Ausgabe	Zutritt gewährt	**Ausgabe**	Zutritt verweigert

(5) Verarbeitung

- Sortiere mit dem Bubble Sort das Array mnr.
- Suche die Mitgliedernummer im Array mnr mit der Binären Suche.
- Ist die Mitgliedernummer gefunden, soll die Suche abbrechen.
- Ist die Mitgliedernummer gefunden, soll am Bildschirm „Zutritt gewährt" ausgegeben werden. Wird die Losnummer nicht gefunden soll „Zutritt verweigert" ausgegeben werden.

(III) Struktogramm



(IV) Programmcode (Python-Code)

Python_II_L2_3_2_2_Binaere_Suche_Mitgliedsnummer.py

```python
#Deklaration und Initialisierung der Eingabewerte
mnr = [1001,1019,1014,1009,1005,1002,1018,1008,1003,1010,1007,1004,1020,1013,1015,1011,1017,1012,1006,1016]
nummer = int(input("Mitgliedsnummer eingeben: "))
laenge = len(mnr)
gefunden = bool(False)

#Verarbeitung
def suche_binaer(gefunden):
    #Ermittle Benachbarte Elemente der Reihe nach und sortiere
    for i in range(1,laenge,+1):
        for j in range(0,laenge-i,+1):
            #Für den Fall das der Wert links > Wert rechts
            if(mnr[j]>mnr[j+1]):
                #Tausche die Werte
                zwischenspeicher = mnr[j]
                mnr[j]=mnr[j+1]
                mnr[j+1]= zwischenspeicher
    #Initialisiere
    gefunden = False
    index_anfang = 0
    index_ende = laenge - 1
    #Solange der Wert nicht gefunden ist und wir nicht am Ende sind
    while(gefunden == False and index_anfang <= index_ende):
        # Ermittle die Mitte
        index_mitte=int((index_anfang+index_ende)/2)
        #Prüfe ob das Element der Mitte mit der Eingabe übereinstimmt
        if(mnr[index_mitte] == nummer):
            gefunden = True
        else:
            #Anderenfalls prüfe ob die Eingabe größer ist als die Mitte
            if(mnr[index_mitte]>nummer):
                #Rücke das Ende um eine Stelle zurück
                index_ende = index_mitte - 1
            else:
                #Rücke den Anfang um eine Stelle vor
                index_anfang = index_mitte + 1
    #Melde ob die Nummer gefunden wurde
    if(gefunden == True):
        print("Zutritt gewährt")
    else:
        print("Zutritt verweigert")


#Verarbeitung: Methodenaufruf
suche_binaer(gefunden)
```

```
#Mitgliedsnummern
mnr = [1001, 1019, 1014, 1009, 1005, 1002, 1018, 1008, 1003, 1010, 1007,
       1004, 1020, 1013, 1015, 1011, 1017, 1012, 1006, 1016]


#Eingabe
nummer = int(input("Mitgliedsnummer eingeben: "))
```

```
laenge = len(mnr)

#Sortieurng mit Bubble Sort
for i in range(1, laenge):
   for j in range(0, laenge-i):
      if mnr[j] > mnr[j+1]:
         zwischenspeicher = mnr[j]
         mnr[j] = mnr[j+1]
         mnr[j+1] = zwischenspeicher
```

```
#Binäre Suche
gefunden = False
index_anfang = 0
index_ende = len(mnr)-1

while gefunden == False and index_anfang <= index_ende:
  index_mitte = int((index_anfang + index_ende) / 2)

  if mnr[index_mitte] == nummer:
    gefunden = True
  elif mnr[index_mitte] > nummer:
    index_ende = index_mitte - 1
  else:
    index_anfang = index_mitte + 1

#Ausgabe
if gefunden:
  print("Zutritt gewährt")
else:
  print("Zutritt verweigert")
```

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L2 3 1 3 Arbeitsauftrag Volleyballspieler suchen

L2_3.1.3 Volleyball – Spieler suchen

(I) Problemstellung

Der Trainer der Abteilung Volleyball des Sportvereins Mühlberger SC möchte eine Erweiterung seiner Software.

Nach der Eingabe eines Spielernamens möchte er eine Information angezeigt bekommen, ob der Spieler im Mannschaftskader ist oder nicht.

Die Namen aller Spieler des Mannschaftskaders sind in dem Array kader erfasst.

kader = ["Armin", "Batu", "Kai", "Sven", "Paul", "Milan", "Goran", "Chris", "Nico",
"Dennis", "Emin", "Luca"]

Der erforderliche Algorithmus soll so optimiert sein, dass die Suche im Array kader beendet wird, sobald der gesuchte Spieler gefunden wurde.

Verwenden Sie für die Implementierung Ihrer Lösung die Datei L2_3_1_3_vorlage_volleyball_spieler_suchen.py, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen L2_3_1_3_volleyball_spieler_suchen.py.

(II) Problemanalyse

(1) Welche Ausgabedaten will man erhalten?

Meldung 'Spieler ist im Mannschaftskader', oder 'Spieler ist nicht im Mannschaftskader'

(2) Welche Eingabedaten werden zur Bearbeitung benötigt?

Name des Spielers

(3) Welche Eigenschaften haben die Eingabe-, Verarbeitungs- und Ausgabedaten? (Variablenliste)

Bedeutung	Typ/Struktur	Variable/Größe
Name des gesuchten Spielers	Zeichenkette	spielername
Merker für Suchergebnis	Boolean	gefunden
Zähler für Schleifenwiederholungen	Ganzzahl	zaehler

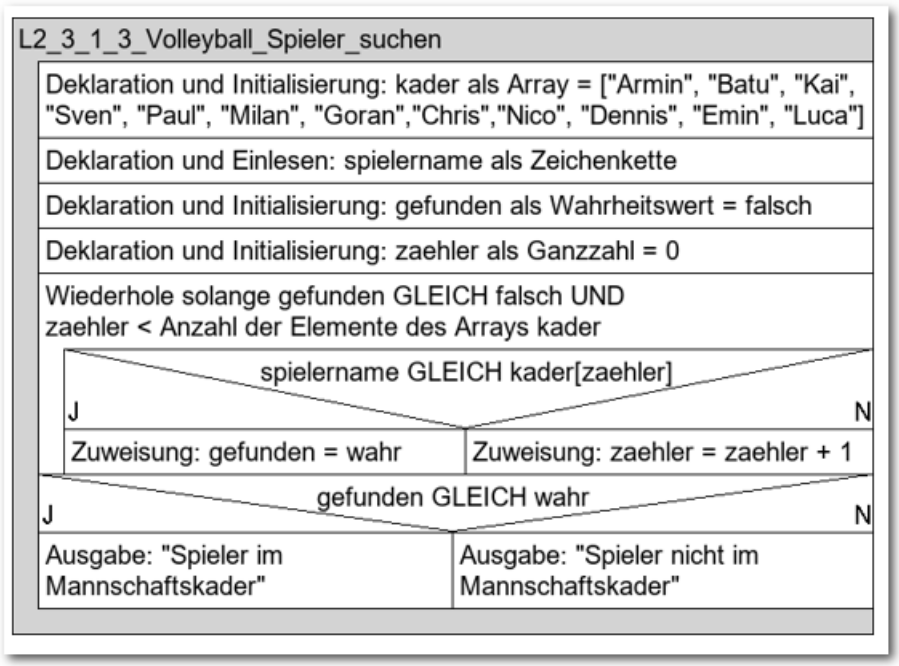
(4) Gewünschter Ablauf des Programms mit Beispieldaten:

Spielername im Kader gefunden		Spielername im Kader nicht gefunden	
Eingabe	Gesuchter Spielername: Batu	**Eingabe**	Gesuchter Spielername: Timo
Ausgabe	Spieler ist im Mannschaftskader	**Ausgabe**	Spieler ist nicht im Mannschaftskader

(5) Verarbeitung

- Suche in jedem Arrayelement nach dem gesuchten Spieler (spielername)
  → Wird der Spieler gefunden, soll die Suche abbrechen.
- Ist der Spieler im Kader soll dies am Bildschirm ausgegeben werden. Wurde er nicht gefunden soll eine entsprechende Meldung ausgegeben werden.

(III) Struktogramm



(IV) Programmcode (Python-Code)

EIGENE LÖSUNG KOMMT!

Python_II_L2_3_1_3_Binaere_Suche_Volleyballspieler_suchen.py

```
2  kader = ["Armin", "Batu", "Kai", "Sven",
3           "Paul", "Milan", "Goran", "Chris", "Nico", "Dennis", "Emin", "Luca"]
4
5
6  #Methode-/Funktionsaufruf
7  def suche_volleyballspieler_binaer():
8      #Deklaration und Initialisierung über die Eingabeaufforderung
9      spielername = input("Gesuchter Spielername: ")
10
11     #Deklaration und Initialisierung eines Wahrheitswertes (Flag)
12     gefunden = False
13
14     #Deklaration und Initialisierung Zähler
15     zaehler = 0
16
17     while(gefunden == False and zaehler < len(kader)):
18         if(spielername == kader[zaehler]):
19             gefunden = True
20         else:
21             zaehler = zaehler + 1
22     if(gefunden):
23         print("Spieler ist im Mannschaftskader")
24     else:
25         print("Spieler ist nicht im Mannschaftskader")
26
27 #Methoden-/Funktionsaufruf
28 suche_volleyballspieler_binaer()
```

Shell

```
>>> %Run Python_II_L2_3_1_3_Binaere_Suche_Volleyballspieler_suchen.py
  Gesuchter Spielername: Chris
  Spieler ist im Mannschaftskader
>>> %Run Python_II_L2_3_1_3_Binaere_Suche_Volleyballspieler_suchen.py
  Gesuchter Spielername: Chrissi
  Spieler ist nicht im Mannschaftskader
```

```
#Musterlöung
kader = ["Armin", "Batu", "Kai", "Sven", "Paul", "Milan", "Goran", "Chris",
"Nico", "Dennis", "Emin", "Luca"]

spielername = input("Gesuchter Spielername: ")
gefunden = False
zaehler = 0

#Suche
while gefunden == False and zaehler < len(kader):
  if spielername == kader[zaehler]:
    gefunden = True
  else:
    zaehler = zaehler + 1

#Ausgabe
if gefunden:
  print("Spieler ist im Mannschaftskader")
else:
  print("Spieler ist nicht im Mannschaftskader")
```

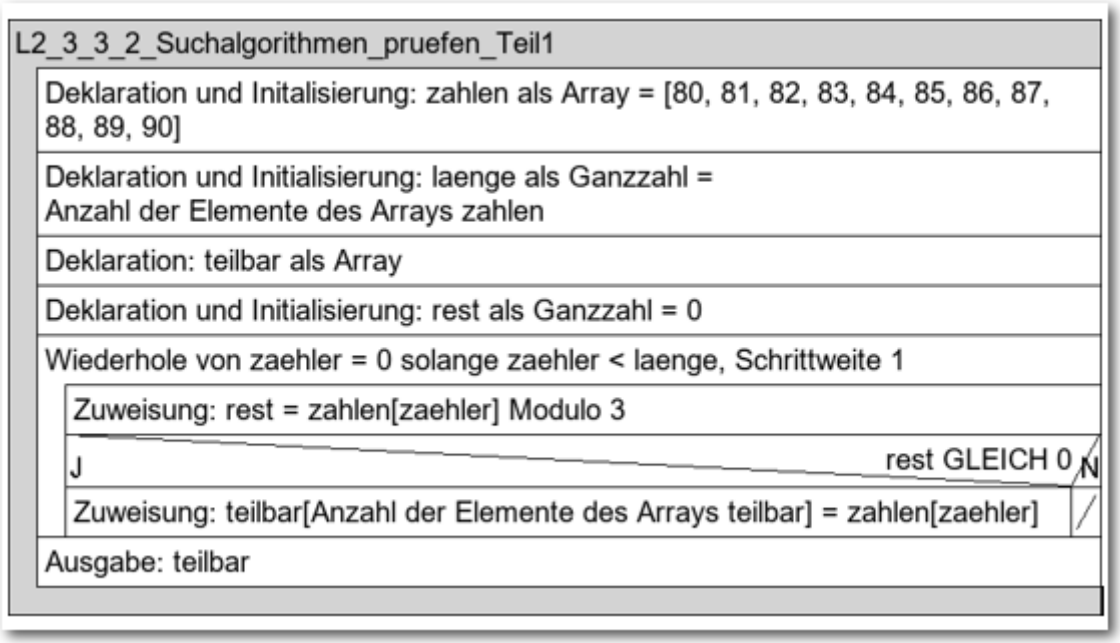
Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L2 3 3 2 Arbeitsauftrag Suchalgorithmen prüfen

L2_3.3.2 Suchalgorithmen prüfen

1 Gegeben ist das Array zahlen, das die natürlichen Zahlen von 80 bis 90 enthält.

zahlen = [80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90]

Mit Hilfe eines Programms soll geprüft werden, welche dieser Zahlen durch 3 teilbar sind. Zur Lösung des beschriebenen Problems wurde bereits folgendes Struktogramm entwickelt:



Hinweis:

Der Ausdruck zahl1 Modulo zahl2 liefert den Rest, den die Division von zahl1 geteilt durch zahl2 ergibt.

Beispiel: 20 Modulo 7 liefert den Wert 6

20 : 7 = 2, Rest 6 (7 * 2 + 6 = 20)

1.1 Führen Sie einen Schreibtischtest durch, indem Sie folgende Tabelle ausfüllen:

zaehler	zahlen[zaehler]	rest	teilbar
0	80	2	[]
1	81	0	[81]
2	82	1	[81]
3	83	2	[81]
4	84	0	[81, 84]
5	85	1	[81, 84]
6	86	2	[81, 84]
7	87	0	[81, 84, 87]
8	88	1	[81, 84, 87]
9	89	2	[81, 84, 87]
10	90	0	[81, 84, 87, 90]

1.2 Implementieren Sie den Programmcode gemäß des abgebildeten Struktogramms.

Für die Modulorechnung wird in den meisten Programmiersprachen das % -Zeichen als Operator verwendet. Der Ausdruck zahl1 Modulo zahl2 wird somit folgendermaßen codiert:

```
zahl1 % zahl2
```

Verwenden Sie für die Implementierung Ihrer Lösung die Datei L2_3_3_2_vorlage_suchalgorithmen_pruefen_teil1.html, die Ihnen im Ordner Aufgaben/Vorlagen in digitaler Form vorliegt.

Speichern Sie Ihre Lösung in Ihrem Ergebnisordner unter dem Namen

L2_3_3_2_suchalgorithmen_pruefen_teil1.html.

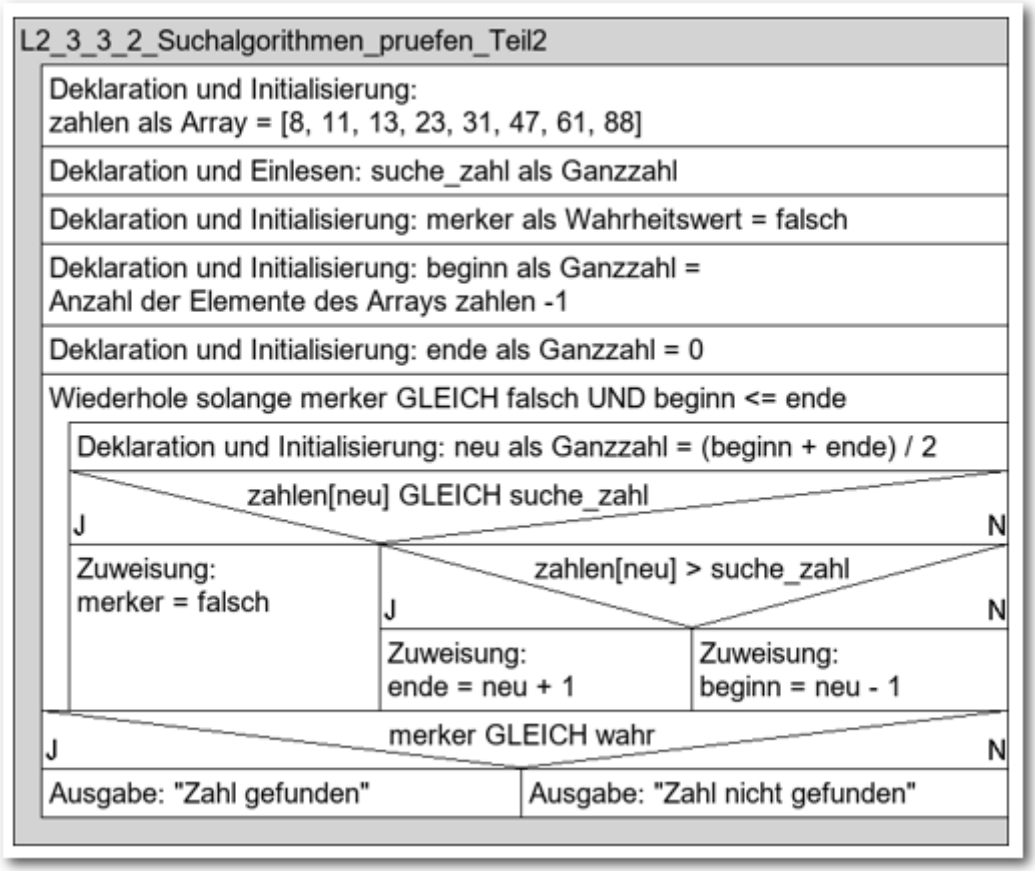
EIGENE LÖSUNG KOMMT!

#Musterlösung

```
zahlen = [80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90]

    laenge = len(zahlen)
    teilbar = []
    rest = 0

    for zaehler in range(laenge):
      rest = zahlen[zaehler] % 3
      if rest == 0:
```

```
        teilbar.append(zahlen[zaehler])

    print(teilbar)
```

2 Ihnen liegt folgendes Struktogramm zur Analyse vor:



2.1 Nennen Sie die Zielsetzung des dargestellten Algorithmus.

Binäre Suche einer Zahl in einem Array.

2.2 Analysieren Sie die einzelnen Anweisungen des dargestellten Algorithmus und lokalisieren Sie vorhandene logische Fehler.

Deklaration und Initialisierung: beginn als Ganzzahl =

Anzahl der Elemente des Arrays zahlen – 1

→ beginn als Ganzzahl = 0

Deklaration und Initialisierung: ende als Ganzzahl = 0

→ ende als Ganzzahl = Anzahl der Elemente des Arrays zahlen - 1

Zuweisung: merker = false

→ merker = true

Zuweisung: ende = neu + 1

→ ende = neu - 1

Zuweisung: beginn = neu – 1

→ ende = neu + 1

7 Dynamische Datenstrukturen – verkettete Liste L3 1

Dynamische Datenstrukturen verkettete Liste L3 1

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L3 1 1 Arbeitsauftrag verkettete Listen

L3_1.1 Dynamische Datenstrukturen: Verkettete Liste

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Aufgabenstellungen das Informations-material

L3_1 Information_verkettete_Liste.docx.

1 Supermarktkasse

In einer Schlange stehen Personen an einer Kasse an. Leni ist die siebte Person in der Schlange und steht zwei Positionen vor Emma. Emma dagegen ist die zweitletzte Person in der Schlange.

Wie viele Personen sind in der Schlange?

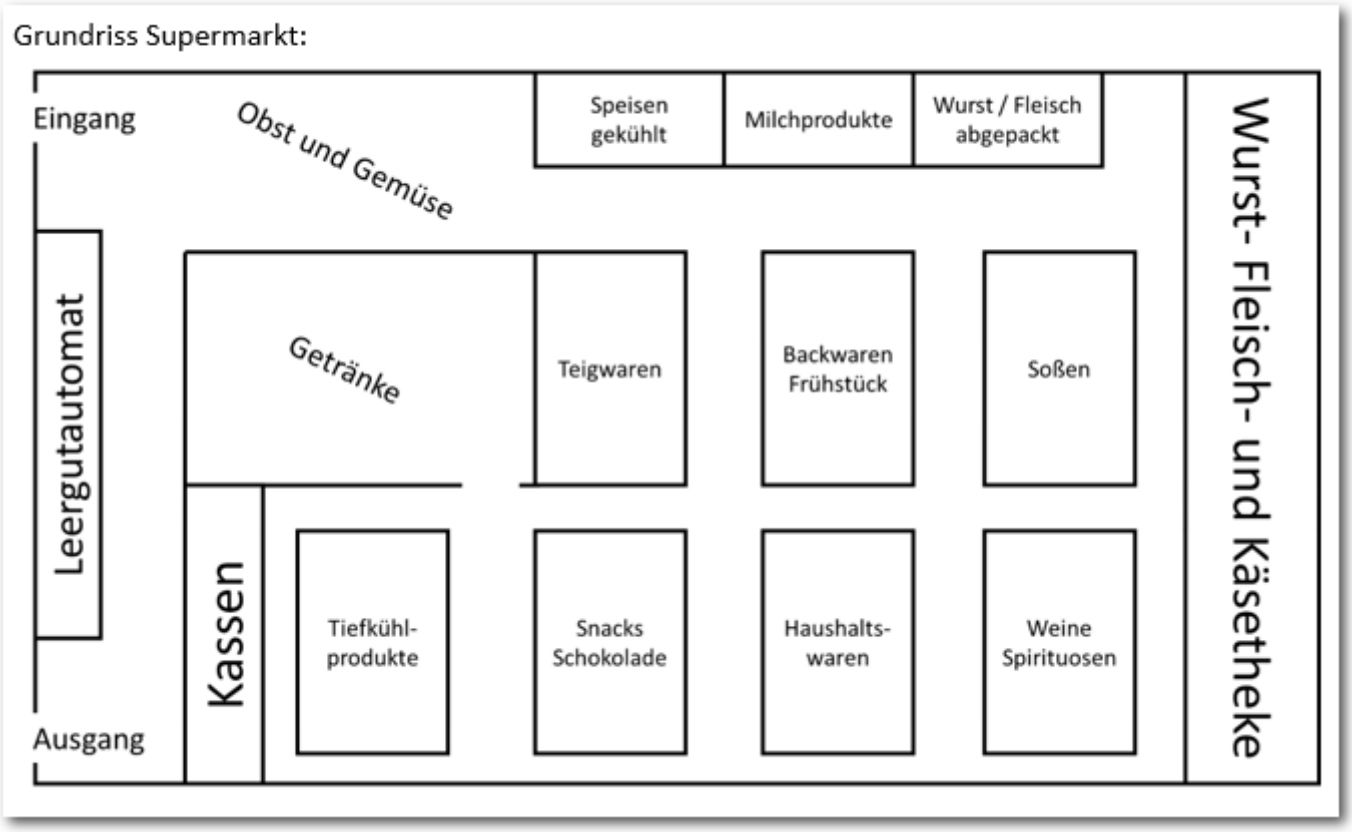
x	x	x	x	x	x	Leni	x	Emma	x

In der Schlange befinden sich 10 Personen.

2 Einkaufsliste Supermarkt

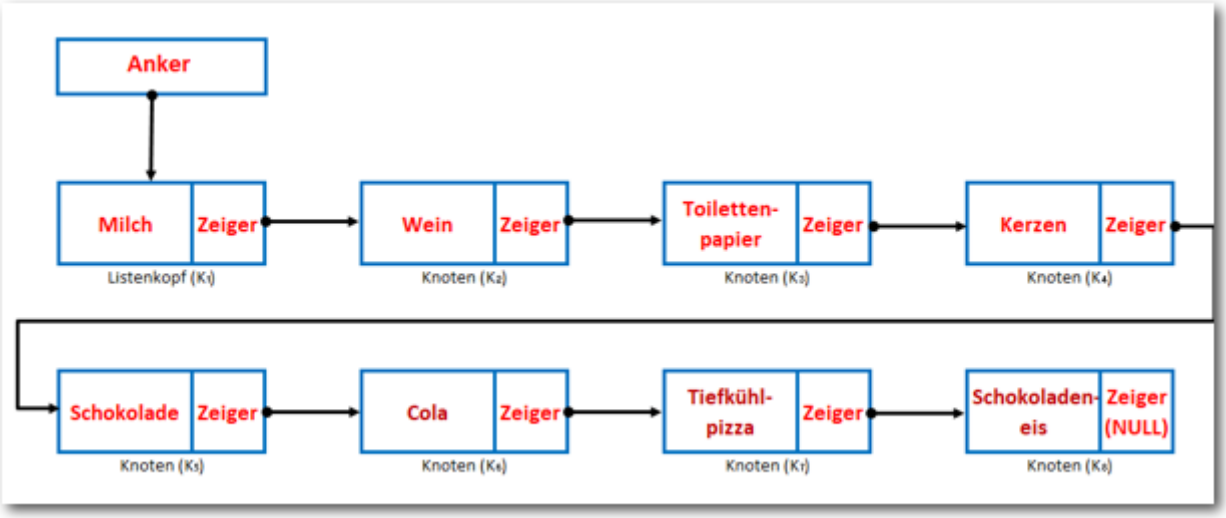
Timo hat seine Freundin Steffi zum Essen eingeladen. Sein Einkaufzettel, mit dem er in den Supermarkt geht, beinhaltet folgende Artikel:

Cola, Wein, Schokoladeneis, Milch, Tiefkühlpizza, Schokolade, Toilettenpapier, Kerzen



2.1 Erstellen Sie einen Einkaufszettel für Timo in Form einer verketteten Liste. Helfen Sie Timo alle Artikel in eine sinnvolle Reihenfolge zu bringen damit er schnell nach Hause kommt, um noch die Wohnung zu putzen.

Individuelle Schülerlösung:



2.2 Timo fällt bei der Fahrt in den Supermarkt auf, dass er noch Leergut zurück bringen kann. Außerdem glaubt er, dass Steffi die Kerzen vielleicht nicht gefallen könnten und streicht sie von der Einkaufsliste.

Überarbeiten Sie die verkettete Liste und beschreiben Sie ausführlich, wie Sie beim Löschen und Hinzufügen von Knoten vorgehen.

"Leergut" hinzufügen:

1. Einen neuen Knoten "Leergut" erzeugen.
2. Den Zeiger des Knotens "Leergut" auf den Knoten "Milch" richten.
3. Den Zeiger des Ankers erhält die Adresse des Knotens "Leergut".

"Kerzen" löschen:

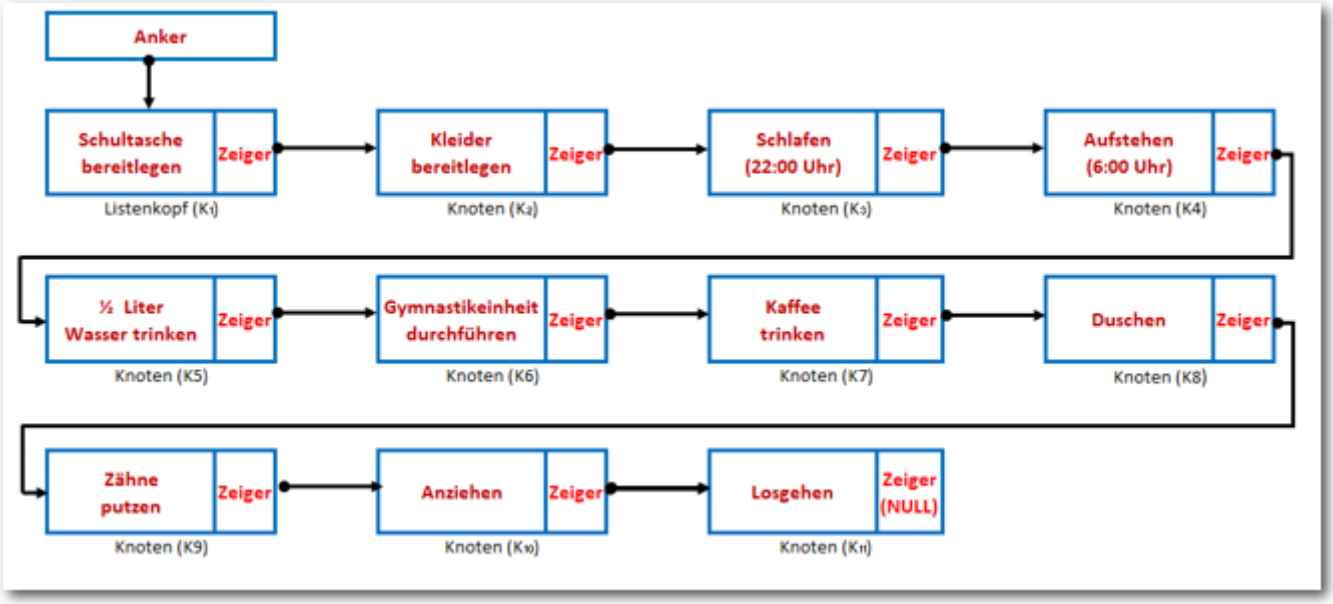
1. Den Zeiger des Knotens "Toilettenpapier" erhält die Adresse des Knotens "Schokolade".

3 Morgenroutine

Um einen guten Start in den Tag zu haben, beschließt Steffi einer Morgenroutine nachzukommen. Die Morgenroutine fängt aber nicht erst am Morgen des Schultages an. Um möglichst wenig Stress am Morgen zu haben, möchte sie ihre Schultasche und die Kleider für den nächsten Tag schon am Vorabend bereitlegen. Steffi möchte um 22 Uhr schlafen gehen, so dass sie um 6:00 Uhr fit ist. Direkt nach dem Aufstehen nimmt sich Steffi vor, einen ½ Liter Wasser zu trinken und eine kurze Gymnastikeinheit durchzuführen. Außerdem muss Steffi noch folgende Tätigkeiten erledigen, bevor sie sich auf den Schulweg macht: Duschen, Zähne putzen, anziehen, Kaffee trinken, losgehen.

3.1. Erstellen Sie für Steffis Morgenroutine eine verkettete Liste in einer sinnvollen Reihenfolge.

Individuelle Schülerlösung:



3.2. Nach der ersten Woche möchte Steffi ihre Morgenroutine anpassen. Weil das Duschen zu viel Zeit am Morgen in Anspruch nimmt, duscht sie von nun an vor dem Schlafengehen. Anstatt wie bisher einen Kaffee zu trinken, möchte Sie auf Tee umsteigen.

Ändern Sie die verkettete Liste und beschreiben Sie genau, wie Sie beim Löschen und Hinzufügen von Knoten vorgehen.

Hinweis: Die Lösung ist auf die Musterlösung aus Aufgabenteil 3.1 bezogen

"Duschen" ändern:

1. Der Zeiger des Knotens "Kaffee trinken" erhält die Adresse des Knotens "Zähne putzen".
2. Der Zeiger des Knotens "Duschen" erhält die Adresse des Knotens "Schlafen".
3. Der Zeiger des Knotens "Kleider bereitlegen" erhält die Adresse des Knotens "Duschen".

"Kaffee trinken" löschen und "Tee trinken" hinzufügen:

1. Einen neuen Knoten "Tee trinken" erzeugen.
2. Der Zeiger des Knotens "Tee trinken" erhält die Adresse des Knotens "Zähne putzen".
3. Den Zeiger des Knotens "Gymnastikeinheit durchführen" erhält die Adresse des Knotens "Tee trinken".

8 Dynamische Datenstrukturen – Stapelspeicher L3 2

Dynamische Datenstrukturen Stapelspeicher L3 2

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L3 2 1 Arbeitsauftrag Stapelspeicher

L3_2.1 Dynamische Datenstrukturen: Stapelspeicher (Stack)

Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Aufgabenstellungen das Informations-material

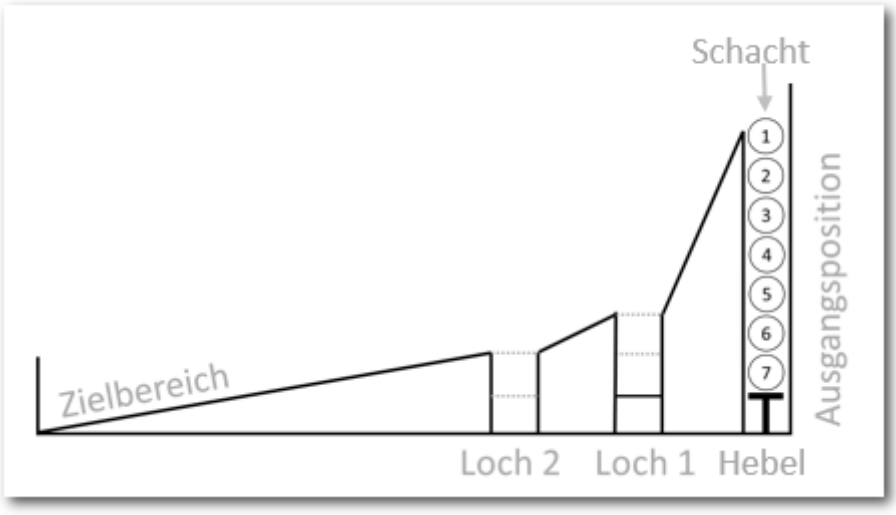
L3_2 Information_Stapelspeicher.docx.

1.1 Murmelbahn – Teil 1

Ihre kleine Schwester hat die Murmelbahn „Die wilde Lifo" zum zweijährigen Geburtstag geschenkt bekommen. Vereinfacht lässt sich die Murmelbahn und das Spielprinzip wie folgt darstellen:

Spielprinzip

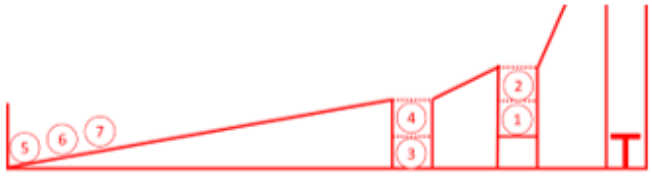
Die Kugeln sind in der Anfangspo-sition in einem Schacht überei-nander gestapelt. Von dort wer-den sie durch einem Hebel in die Murmelbahn gedrückt. In der Bahn befinden sich zwei Löcher, in die jeweils zwei Kugeln passen. Ist ein Loch voller Kugeln, rollt die nächste Kugel über das Loch. Das Spiel endet, wenn sich keine Ku-geln mehr in der Anfangsposition befinden.



Aufgabe

Benennen Sie die Kugeln in der Reihenfolge, in der sie im Zielbereich ankommen. Verwenden Sie zur Dokumentation Ihrer Überlegungen folgende Tabelle.

Schritt	Ausgangsposition	Loch 1	Loch 2	Zielbereich
	1, 2, 3, 4, 5, 6, 7			
1	2, 3, 4, 5, 6, 7	1		
2	3, 4, 5, 6, 7	1, 2		
3	4, 5, 6, 7	1, 2	3	
4	5, 6, 7	1, 2	3, 4	
5	6, 7	1, 2	3, 4	5
6	7	1, 2	3, 4	5, 6
7		1, 2	3, 4	5, 6, 7

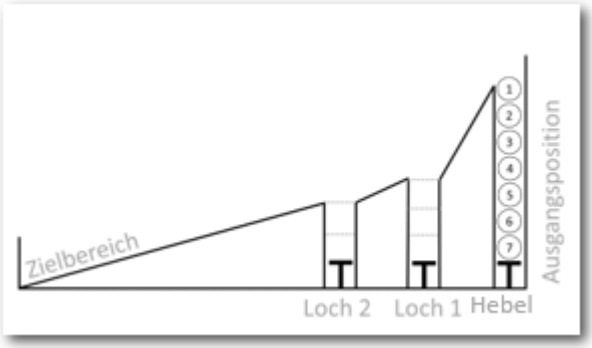


1.2 Murmelbahn – Teil 2

Nachdem Ihre Schwester viel Spaß mit der Murmelbahn hatte, entscheiden Sie sich, ihr die Weiter-entwicklung des Spiels, „Die noch wildere Lifo", zu schenken. Vereinfacht lässt sich die Murmelbahn und das Spielprinzip wie folgt darstellen:

Spielprinzip

Die Kugeln werden von der Ausgangsposition nacheinan-der durch einem Hebel in die Murmelbahn gedrückt. In der Bahn befinden sich zwei Löcher. In das erste Loch passen drei Kugeln, in das zweite Loch zwei Kugeln. Ist eines der Löcher voll, werden alle Kugeln aus diesem Loch nacheinander in die Murmelbahn zurück gedrückt. Erst wenn dieses Loch wieder leer ist, werden die Kugeln aus dem Schacht der Ausgangssituation in die Kugelbahn gedrückt, oder das andere volle Loch geleert.



Ansonsten gelten die bereits bekannten Regeln aus der Murmelbahn Teil 1.

Aufgabe

1.2.1 Stellen Sie den Spielverlauf in der nachfolgenden Tabelle dar.

Schritt	Ausgangsposition	Loch 1	Loch 2	Zielbereich
	1, 2, 3, 4, 5, 6, 7			
1	2, 3, 4, 5, 6, 7	1		
2	3, 4, 5, 6, 7	1, 2		
3	4, 5, 6, 7	1, 2, 3		
4	4, 5, 6, 7	1, 2	3	
5	4, 5, 6, 7	1	3, 2	
6	4, 5, 6, 7		3, 2	1
7	4, 5, 6, 7		3	1, 2
8	4, 5, 6, 7			1, 2, 3
9	5, 6, 7	4		1, 2, 3
10	6, 7	4, 5		1, 2, 3
11	7	4, 5, 6		1, 2, 3
12	7	4, 5	6	1, 2, 3
13	7	4	6, 5	1, 2, 3
14	7		6, 5	1, 2, 3, 4
15	7		6	1, 2, 3, 5
16	7			1, 2, 3, 4, 5, 6
17		7		1, 2, 3, 4, 5, 6

1.2.2 Nennen Sie die Ziffer der Kugel, die als erstes im Zielbereich ankommt und begründen Sie Ihre Entscheidung.

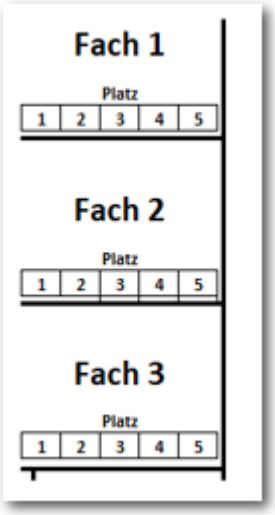
Kugel 1 ist die erste Kugel im Zielbereich. Im Schritt 3 ist das erste Loch voll und die Kugel 1 liegt auf

der untersten Ebene. Nach den Spielregeln wird nun das erste Loch vollständig ge-leert. Da in Schritt 6 das zweite Loch voll ist, rollt die letzte Kugel aus dem ersten Loch als erstes in den Zielbereich.

2 Lebensmittelmarkt SuperSpar

Der Lebensmittelmarkt SuperSpar kauft von verschiedenen Großhändlern stark verbilligte Restposten ein. So kann der Lebensmittelmarkt seinen Kunden günstige Preise anbieten und trotzdem Gewinne erzielen.

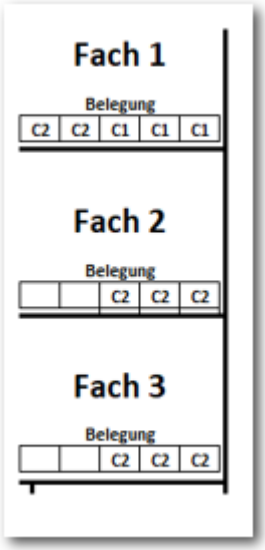
Die Ladenregale im Super-Spar verfügen über drei Fächer. In jedem Fach ist Platz für fünf Waren, die hinter-einander gestellt werden. Kommt neue Ware, wird diese von vorne aufgefüllt und die bereits vor-handene Ware wird nach hin-ten geschoben (Lifo-Prinzip). Sobald das erste Fach voll ist, werden die Waren in das zweite Fach nach dem Lifo-Prinzip gestellt.



Beispiel:

Am Montag kommt neue Ware im SuperSpar an. Es werden drei Flaschen Cola (C1) in das Ladenregal geräumt. Am Diens-tag kommt die zweite Liefe-rung mit zehn Flaschen Cola (C2).

Am Mittwoch entnimmt ein Kunde zwei Flaschen Cola aus dem Fach 2.



2.1 Der SuperSpar erhält am Montagmorgen eine Lieferung von fünf Sweeties (S1) und räumt diese in das Fach 1 des Regals ein. Am gleichen Tag kaufen die Kunden vier Sweeties.

Stellen Sie das Fach 1 nach der Befüllung mit den fünf Sweeties (S1) und nach Ladenschluss dar.

Nach der Befüllung Nach Ladenschluss

2.2 Am Dienstagmorgen kommt eine weitere Lieferung von zehn Sweeties (S2). Die Süßigkeiten wer-den in das Regal eingeräumt. Begonnen wird im Fach 1, sofern dort noch Platz vorhanden ist.

Ist dieses voll, wird Fach 2 befüllt, anschließend Fach 3.

Im Laufe des Tages kaufen Kunden drei Sweeties aus Fach 1 und vier Sweeties aus Fach 2.

Stellen Sie das Regal nach der Befüllung der weiteren zehn Sweeties (S2) und nach dem La-den-schluss dar.

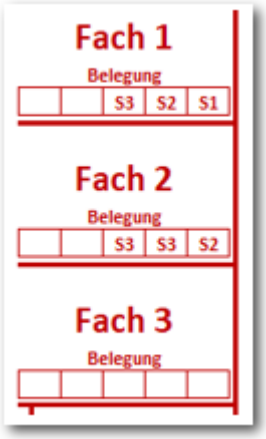
Nach der Befüllung Nach Ladenschluss

2.3 Am Mittwochmorgen kommt erneut eine Lieferung von acht Sweeties (S3), die in das Regal einge-räumt werden. (Reihenfolge des Einräumens wie in Aufgabe b) beschrieben.)

Aus jedem Fach werden am Mittwoch zwei Sweeties gekauft.

Wie viele Sweeties aus der ersten (S1), aus der zweiten (S2) bzw. aus der dritten (S3) Liefe-rung sind noch im Regal?

Nach der Befüllung Nach Ladenschluss



Es sind noch ein Sweety aus der ersten Lieferung (S1), zwei Sweeties aus der zweiten Lie-ferung (S2) und drei Sweeties aus der dritten Lieferung (S3) im Regal.

3 Internetrecherche

Das Internet bietet einen breiten Fundus an Informationen und wird deswegen gerne für die Recher-che benutzt. Wechseln Sie von der Internetseite A auf die Internetseite B wird die Adresse der Inter-netseite A in einen Stapelspeicher gelegt. Klicken Sie nun auf das „Zurück"-Icon, wird das vorherige Element aus dem Stack aufgerufen und Sie kehren zur Internetseite A zurück.

Der Browserverlauf eines Nutzers ist nachfolgend abgebildet.

Zeile	Adresse
1	**PUSH**("https://www.youtube.com/results?search_query=stapelspeicher")
2	**PUSH**("https://de.wikipedia.org/wiki/Stapelspeicher")
3	**POP**()
4	**PUSH**("http://deacademic.com/dic.nsf/dewiki/1323671")
5	**PUSH**("https://www.heise.de/")
6	**PUSH**("https://www.wim.uni-mannheim.de/de/informatik-und-wirtschaftsinformatik/")
7	**PUSH**("https://www.informatik.kit.edu/")
8	**PUSH**("http://www.warum-informatik.de")
9	**POP**()
10	**POP**()

3.1 Auf welcher Internetseite befindet sich der Nutzer derzeit?

Das POP() in Zeile 9 ruft die Internetseite "https://www.informatik.kit.edu/" erneut auf.

Der Benutzer ist auf der Internetseite " https://www.wim.uni-mannheim.de/de/informatik-und-wirt-schaftsinformatik/", da das POP() in Zeile 10 diese Internetseite wieder aufruft.

3.2 Auf welchen Internetseiten war der Nutzer zweimal?

Durch die drei POP()-Befehle werden folgende Webseiten noch einmal aufgerufen:

- https://www.youtube.com/results?search_query=stapelspeicher
- https://www.wim.uni-mannheim.de/de/informatik-und-wirtschaftsinformatik/
- https://www.informatik.kit.edu/

9 Dynamische Datenstrukturen – Warteschlange L3 3

Dynamische Datenstrukturen Warteschlange L3 3

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L3 3 1 Arbeitsauftrag Warteschlange

L3_3.1 Dynamische Datenstrukturen: Warteschlange (Queue)

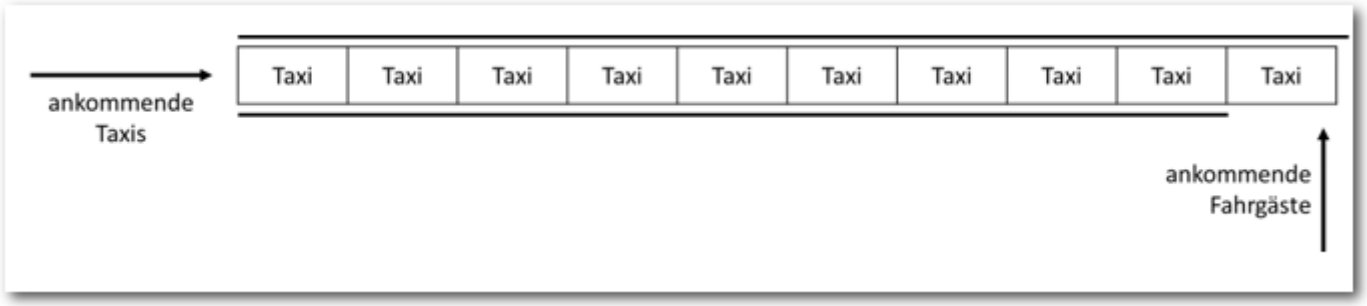
Hinweis:

Beachten Sie zur Bearbeitung der nachfolgenden Aufgabenstellungen das Informations-material

L3_3 Information_Warteschlange.docx.

1 Taxistand – Teil 1

Am Flughafen gibt es einen Taxistand mit zehn Stellplätzen. Ankommende Taxis stellen sich in einer Schlange an. Im Taxistand kann nicht gewendet werden. Deswegen kann immer nur das vorderste Taxi Fahrgäste aufnehmen und abfahren. Das folgende Schaubild soll den Sachverhalt verdeutlichen.



Ankommende Taxis werden von einem elektronischen Erfassungssystem mit der Taxinummer erfasst. Verlässt ein Taxi den Taxistand wird dies ebenfalls erfasst. Ihnen liegen folgende Daten vor:

Zeit	Taxinummer	Ausfahrt/Einfahrt
14:01	T254	Einfahrt
14:01	T255	Einfahrt
14:02	T254	Ausfahrt
14:05	T432	Einfahrt
14:07	T432	Ausfahrt
14:10	T123	Einfahrt

Hinweis:

Kommen mehrere Taxis zur gleichen Zeit an (vgl. 14:01 Uhr), sind sie in der Tabelle nach ihrer Ankunft aufgelistet (T254 vor T255).

1.1 Aus der Auflistung wird ersichtlich, dass im automatischen Erfassungssystem ein Fehler sein muss. Beschreiben Sie diesen Fehler.

Der Taxistand ist nach dem FIFO-Prinzip aufgebaut. Das Taxi T254 fährt als erstes Taxi in den Taxistand ein und auch wieder als erstes Taxi aus dem Taxistand. T432 fährt als drittes Taxi in den Taxistand aber als zweites Taxi aus dem Taxistand. Dies widerspricht dem FIFO-Prinzip und ist somit ein Fehler.

Sie erhalten neue Daten vom 01.08. aus dem automatischen Erfassungssystem.

Zeit	Taxinummer	Ausfahrt/Einfahrt
08:00	T021	Einfahrt
08:02	T099	Einfahrt
08:05	T085	Ausfahrt
08:08	T432	Ausfahrt
08:10	T021	Ausfahrt
08:12	T123	Einfahrt
08:16	T099	Ausfahrt
08:18	T525	Einfahrt
08:21	T456	Einfahrt

1.2 Wie viele Taxis befanden sich vor der Einfahrt des Taxis T021 im Taxistand?

Ausgehend vom FIFO-Prinzip fährt das Taxi zuerst aus dem Taxistand, das auch zuerst in den Taxistand hinein gefahren ist. Folglich müssen alle Taxis, die vor dem Taxi T021 aus dem Taxistand fahren bei des-

sen Einfahrt schon im Taxistand gewesen sein. Die Taxis T085 und T432 fahren vor dem Taxi T021 aus dem Taxistand. Es waren also zwei Taxis im Taxistand, als das Taxi T021 einfuhr.

1.3 Welche Taxis befinden sich in welcher Reihenfolge um 08:22 Uhr im Taxistand? Vervollständigen Sie zur Beantwortung der Frage die folgende Tabelle.

	10	9	8	7	6	5	4	3	2	1
08:00								T021	T432	T085
08:02							T099	T021	T432	T085
08:05								T099	T021	T432
08:08									T099	T021
08:10										T099
08:12									T123	T099
08:16										T123
08:18									T525	T123
08:21								T456	T525	T123

Um 08:22 befinden sich die Taxis T123, T525 und T456 im Taxistand.

1.4 Um 08:01 Uhr kommt Sandra Meier am Taxistand an. Es warten keine Fahrgäste auf ein Taxi. Sandra Meier möchte aber ausschließlich mit dem Taxi T021 fahren. Wann ist Sandra Meier abgefahren und wie viele Fahrgäste hat sie vorgelassen?

Sandra Meier fährt um 08:10 ab. Sie lässt zwei Personen vor (T085 und T432).

1.5 In der Zeit von 08:30 bis 09:00 arbeitete das automatische Erfassungssystem fehlerhaft. Es wurden zwar die Uhrzeiten und die Taxinummern erfasst, jedoch nicht, ob es sich um eine Ausfahrt oder Einfahrt handelte.

Vervollständigen Sie die nachfolgenden Lücken unter Beachtung der dynamischen Datenstruktur Warteschlange.

Zeit	Taxinummer	Ausfahrt/Einfahrt
08:30	T050	Einfahrt
08:32	T599	Einfahrt
08:33	T123	**Ausfahrt**
08:35	T525	**Ausfahrt**
08:38	T456	**Ausfahrt**
08:40	T050	**Ausfahrt**
08:42	T321	**Einfahrt**
08:46	T099	**Einfahrt**
08:48	T599	**Ausfahrt**
08:51	T321	**Ausfahrt**
08:53	T555	**Einfahrt**
08:56	T099	**Ausfahrt**

Beachte:

Um 08:30 befanden sich am Taxistand noch die Taxis

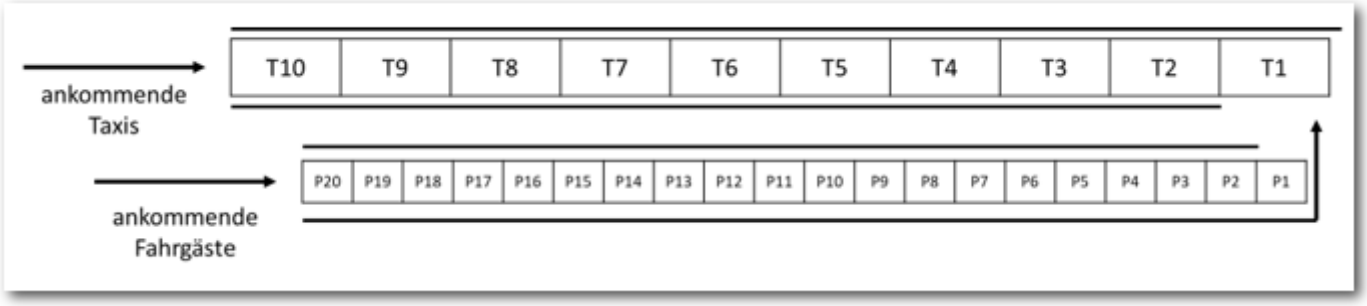
08:12 T123 Einfahrt

08:18 T525 Einfahrt

08:21 T456 Einfahrt

2 Taxistand – Teil 2

Aufgrund einiger Unstimmigkeiten der Fahrgäste bei der Taxivergabe hat der Flughafen neben dem Taxistand auch eine Wartelinie für Fahrgäste eingerichtet (P1 – P20).



Als zusätzliche Information können zukünftig die Sitzplätze eines Taxis gespeichert werden. So hat das Taxi T432.3 bspw. 3 Sitzplätze, das Taxi T009.6 hat 6 Sitzplätze. Außerdem muss auch die Anzahl der zusammengehörigen Fahrgäste (= Fahrgastgruppe) beachtet werden. So hat die Fahrgastgruppe F1.2 beispielsweise 2 Personen und die Fahrgastgruppe F2.8 sogar 8 Personen.

Zeit	Taxinummer	Ausfahrt/Einfahrt
08:00	T021.5	Einfahrt
08:02	T021.5	Ausfahrt

Zeit	Ankommende Fahrgastgruppen
08:01	F1.3
08:01	F2.2

In diesem Beispiel kommt um 08:00 das Taxi T021.5 an. Um 08:01 kommen die Fahrgastgruppen F1 mit drei Personen und die Fahrgastgruppe F2 mit zwei Personen. Die Fahrgastgruppe F1.3 kann direkt abfahren, sodass um 08:02 nur noch die Fahrgastgruppe F2.2 auf ein Taxi wartet. Folglich können mehrere Fahrgastgruppen nicht mit einem Taxi fahren (z. B. aufgrund unterschiedlicher Fahrtziele).

Eine Fahrgastgruppe kann sich dagegen auf zwei oder mehrere Taxis verteilen.

Die nachfolgende Tabelle verdeutlicht den Sachverhalt.

Taxi	....		5		4		3		2		1	
Person	...	...	10	9	8	7	6	5	4	3	2	1
08:00	....										T021.5	
	...	...										
08:01	....										T021.5	
	...	...						F2.2	F2.2	F1.3	F1.3	F1.3
08:02	....											
	...	...									F2.2	F2.2

Sie erhalten aus dem automatischen Erfassungssystem neue Daten vom 02.08.

Zeit	Taxinummer	Ausfahrt/Einfahrt
08:00	T021.5	Einfahrt
08:02	T009.6	Einfahrt
08:07	T021.5	Ausfahrt
08:09	T009.6	Ausfahrt
08:11	T010.6	Einfahrt
08:13	T010.6	Ausfahrt
08:14	T432.3 T258.3 T554.5	Einfahrt Einfahrt Einfahrt
08:16	T432.3	Ausfahrt
08:18	T599.3	Einfahrt

Zeit	Ankommende Fahrgastgruppen
08:02	F1.3
08:02	F2.9
08:09	F3.1
08:12	F4.2

2.1 Vervollständigen Sie die nachfolgende Tabelle mit den Plätzen im Taxistand und der Warteschlange für die Fahrgastgruppen. Beachten Sie die Regeln für die dynamische Datenstruktur Warteschlange.

08:00	....																	T021.5	
	...	...																	
08:02	....																T009.6		T021.5	
	...	...	F2.9	F2.9	F2.9	F2.9	F2.9	F2.9	F2.9	F2.9	F2.9	F1.3	F1.3	F1.3						
08:07	....																	T009.6	
	...	...				F2.9	F2.9	F2.9	F2.9	F2.9	F2.9	F2.9	F2.9	F2.9						
08:09	....																		
	...	...									F3.1	F2.9	F2.9	F2.9						
08:11	....																	T010.6	
	...	...									F3.1	F2.9	F2.9	F2.9						
08:12	....																	T010.6	
	...	...							F4.2	F4.2	F3.1	F2.9	F2.9	F2.9						
08:13	....																		
	...	...										F4.2	F4.2	F3.1						
08:14	....								T554.5		T258.3		T432.3						
	...	...										F4.2	F4.2	F3.1						
08:16	....										T554.5		T258.3						
	...	...											F4.2	F4.2						
08:18	....								T599.3		T554.5		T258.3						
	...	...											F4.2	F4.2						

2.2 Wie viele Taxis und wie viele Fahrgäste sind um 08:06 Uhr im Taxistand?

Taxis: T021.5 + T009.6 = 2 Taxis

Fahrgäste: 3 x F1.3 + 9 x F2.9 = 12 Fahrgäste

2.3 Ermitteln Sie die Fahrgastzahl des Taxis T010.6, wenn es um 08:13 Uhr den Taxistand verlässt.

Es sind drei Fahrgäste im Taxi. Die restlichen 6 Fahrgäste der Fahrgastgruppe F2 sind bereits mit dem Taxi T009.6 abgefahren.

2.4 Wie viele Fahrgäste können mindestens und höchstens bis 08:30 Uhr abfahren, wenn kein weiteres Taxi mehr kommt?

Es warten noch zwei Fahrgäste, die mit dem Taxi T258.3 abfahren. Daneben stehen noch zwei Taxis mit maximal 8 (5+3) Plätzen in der Warteschlange. Kommen zwei Fahrgastgruppen mit jeweils nur einer Person, so können mindestens vier Personen abfahren. Kommt eine Fahrgastgruppe mit 5 und eine Fahrgastgruppe mit 3 Personen, so können höchstens zehn Personen abfahren.

10 Dynamische Datenstrukturen – Baum L3 4

Dynamische Datenstrukturen
Baum L3 4

Thema:	Grundlagen der Programmierung in Python – Arbeitsauftrag Quelle: L3 4 1 Arbeitsauftrag Baum

L3_4.1 Dynamische Datenstrukturen: Baum

Hinweis:

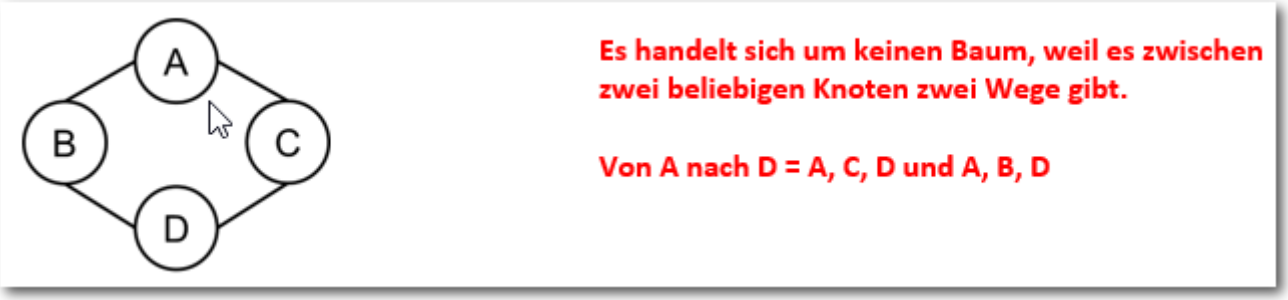
Beachten Sie zur Bearbeitung der nachfolgenden Aufgabenstellungen das Informations-material L3_4 Information_Baum.docx.

1 Nennen Sie Alltagsbeispiele für die Datenstruktur Baum.

Familienstammbuch, Organigramm, Datei-Struktur im Rechner

2 Begründen Sie, ob es sich beim nachfolgenden Schaubild um einen Baum handelt.

Es handelt sich um keinen Baum, weil es zwischen zwei beliebigen Knoten zwei Wege gibt. Von A nach D = A, C, D und A, B, D



3 Markieren Sie einen Teilbaum im nachfolgenden Baum.

4 Beurteilen Sie, ob der nachfolgende Baum vollständig, voll und/oder geordnet ist.

GEORDNET:

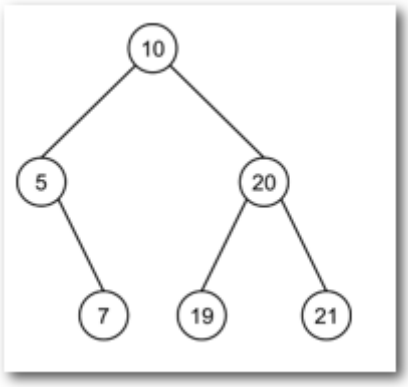
Nein, zwar sind alle Knoten links vom Knoten 10 kleiner und alle rechts sind größer. Aber der Knoten 5 hat nur ein „rechtes" Kind.

VOLL:

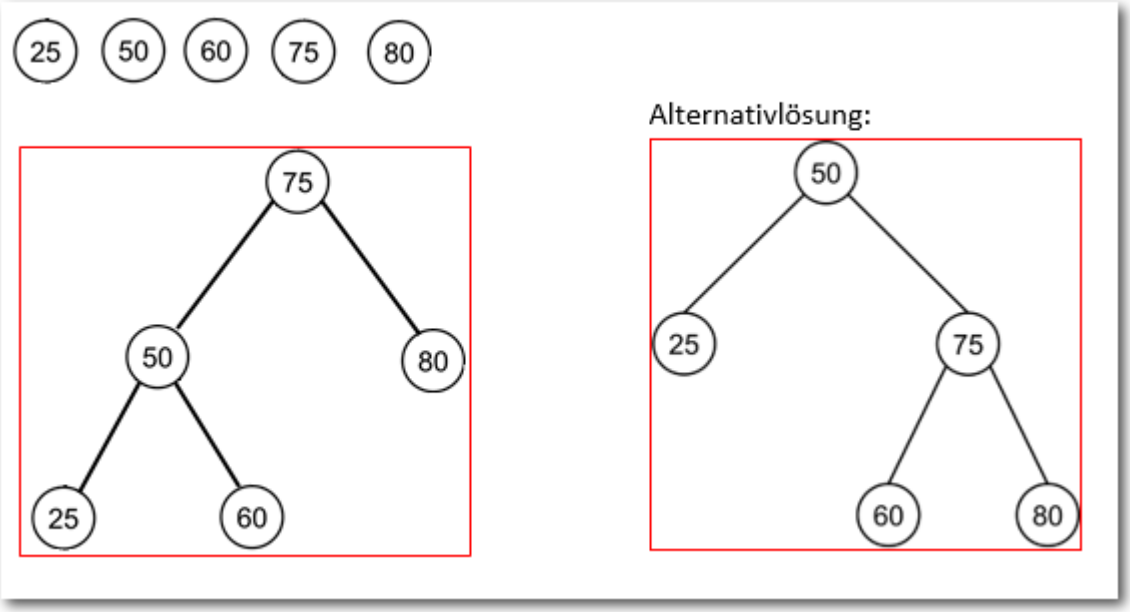
Nein, der Knoten 5 ist kein Blatt und besitzt nur ein Kind (Knoten 7)

VOLLSTÄNDIG:

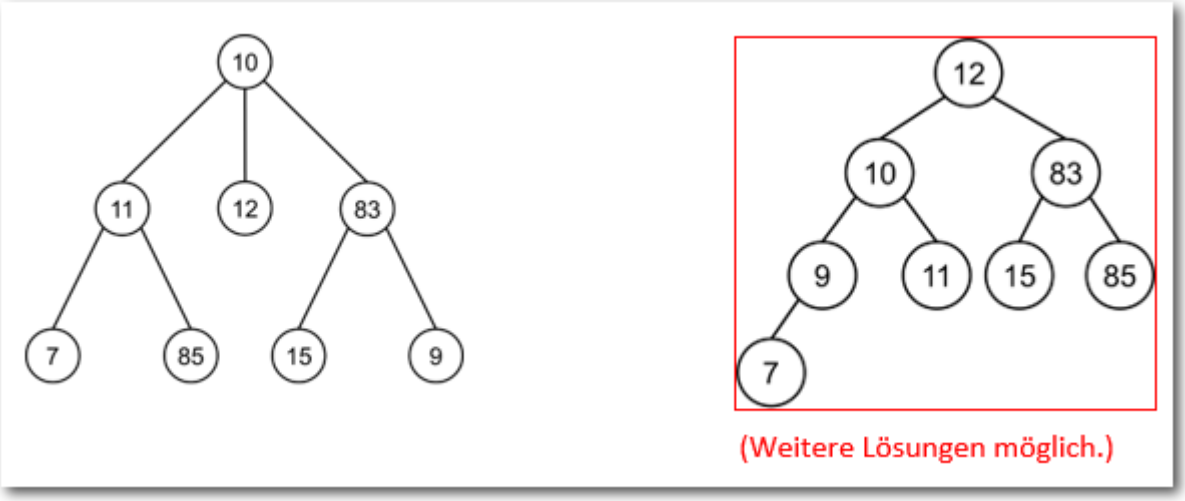
Ja, weil alle Blätter auf einer Höhe sind. Knoten 5 ist kein Blatt, weil er in 7 einen Nachfolger hat.



5 Überführen Sie die Knoten {25, 50, 60, 75, 80} in einen vollen, geordneten Binär baum.


25
50
60
75
80
75
50
80
25
60
Alternativlösung:
50
25
75
60
80

6 Überführen Sie den folgenden Baum in einen Binärbaum.



7 Fügen Sie nacheinander die Knoten {B, A} in den gegebenen Binärbaum ein. Achten Sie darauf, dass der Binärbaum weiterhin geordnet ist.