Aufgabe 1

Erkläre, was ein *abstrakter Datentyp* ist und gib ein Beispiel dafür an.

Aufgabe 2

Beschreibe die Datenstruktur *Array* und nenne Anwendungen dafür.

Aufgabe 3

Gegeben ist ein Array. Nenne neben dem Lese- und Schreibzugriff auf seine einzelnen Elemente noch weitere wünschenswerte Operationen auf Arrays.

Aufgabe 4

(a) Wie wird das Funktionsprinzip von Stacks oft abgekürzt?

(b) Zähle Standardoperationen für Stacks auf.

(c) Nenne zwei Anwendungen für Stacks.

Aufgabe 5

Vervollständige die Methoden der Python-Klasse `Stack`.

```
class Stack:

    def __init__(self):
        self.items = []

    def push(self, item):
        ...

    def pop(self):
        ...

    def is_empty(self):
        ...

    def size(self):
        ...
```

Aufgabe 6

Stelle den Stack `S` beim Ausführen folgender Operationen als Liste dar, die von links nach rechts wächst. Zu Beginn ist der Stack leer.

```
S = Stack()
S.push(3)
S.push(5)
x = S.pop()
S.push(1)
S.push(x)
S.push(4)
```

Aufgabe 7

Erkläre, wie die Funktion `check(string)` mit Hilfe des Stacks `S` eine Zeichenketten auf korrekte Klammerung prüft.

```
from stack import Stack
def check(string):
    S = Stack()
    for c in string:
        if c == '(':
            S.push(c)
        elif c == ')':
            if S.is_empty():
                return False
            else:
                S.pop()
        else:
            pass
    if S.is_empty():
        return True
    return False
```

Aufgabe 8

(a) Wie wird das Funktionsprinzip von Warteschlangen abgekürzt?

(b) Zähle Standardoperationen für Warteschlangen auf.

(c) Nenne zwei Anwendungen für Warteschlangen.

Aufgabe 9

Vervollständige die Methoden der Python-Klasse `Queue`.

```
class Queue:

    def __init__(self):
        self.items = []

    def enqueue(self, item):
        ...

    def dequeue(self):
        ...

    def is_empty(self):
        ...

    def size(self):
        ...
```

Warum ist eine Implementierung mit Listen nicht effizient?

Aufgabe 10

Stelle die Warteschlange `Q` beim Ausführen folgender Operationen als Liste dar, wenn die Elemente links eingefügt und rechts entfernt werden. Zu Beginn ist `Q` leer.

```
Q = Queue()
Q.enqueue(5)
Q.enqueue(1)
Q.enqueue(4)
x = Q.dequeue()
Q.enqueue(3)
Q.enqueue(x)
y = Q.dequeue()
```

Aufgabe 11

Erläutere den Aufbau einfach verketteter Listen (EVL) anhand einer Skizze und beschreibe, wie man ...

- ein Element am Anfang einer EVL hinzugefügt,
- ein Element vom Anfang einer EVL entfernt,
- eine EVL durchläuft (z. B. nach einem Wert sucht oder zählt).

Aufgabe 12

Nenne drei verschiedene Anwendungen von verketteten Listen.

Aufgabe 13

Vervollständige die Methoden der Python-Klasse `Linkedlist`, indem du den leeren Knoten als leere Liste `[]` und Listen der Form `[value, reference]` als Container für den zu speichernden Wert die Referenz auf den nächsten Knoten implementierst.

```
class Linkedlist:
    '''Klasse für einfach verkettete Listen (EVL)'''

    def __init__(self):
        self.start = []

    def insert(self, item):
        '''Füge item am Anfang der EVL ein.'''
        ...

    def is_empty(self):
        '''Ist EVL leer, gib True zurück; sonst False'''
        ...

    def remove(self):
        '''Entferne 1. Knoten und gib den Wert zurück'''
        '''(Ist die EVL leer, gib None zurück)'''
        ...
```

Aufgabe 14

Stelle die einfach Verkettete Liste `EVL` beim Ausführen folgender Operationen als Verkettung von Listen der Länge 2 und der Länge 0 dar, wobei neue Knoten links eingefügt und entfernt werden.

```
EVL = Linkedlist()
EVL.insert(7)
EVL.insert(3)
x = EVL.remove()
EVL.insert(2)
print(x)
```

Aufgabe 15

Stelle die im Speicherabbild enthaltene Daten einer einfach verkettete Liste mit der Startadresse `0x14` sequentiell dar.

- Der Speicher ist linear aufgebaut (auf die Adresse `0x0F` folgt `0x10`). Aus Platzgründen erfolgt die Darstellung mehrzeilig. Die Adressierung der Daten erfolgt durch eine zweistellige Hexadezimalzahl, wobei die linke Ziffer die Zeilennumer und die rechte Ziffer Spaltennumer darstellt.
- Ein *Knoten* besteht aus zwei benachbarten Speicherzellen. Die erste Speicherzelle enthält einen hexadezimalen Datenwert, die zweite die eine Adresse.
- `0x00` steht für den Nullzeiger oder eine leere Speicherzelle.
- Der Speicher kann auch noch andere oder verwaiste Daten enthalten.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0		00	0B	41	00	00	00	00	00	00	00	00	00	00	00	22
1	37	00	00	00	3A	0F	00	32	41	00	00	00	00	00	00	00
2	00	00	00	00	00	00	00	00	00	0C	14	17	00	00	46	20
3	00	00	0A	22	00	00	00	32	00	00	00	00	00	24	46	00
4	00	00	00	00	00	00	00	02	04	00	00	00	00	00	00	00

Aufgabe 16

Was ist ein ungerichteter Graph?

Aufgabe 17

Was ist ein gerichteter Graph?

Aufgabe 18

Stelle den Graphen G mit

$V = \{a, b, c, d, e\}$

und

$E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{b, d\}, \{d, d\}\}$.

graphisch dar.

Aufgabe 19

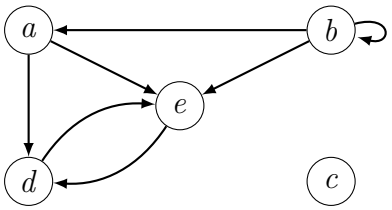
Stelle den Graphen $G = (V, E)$ mit
$V = \{a, b, c, d, e\}$
und
$E = \{(a, c), (b, d), (c, e), (d, c), (e, c)\}$
graphisch dar.

Aufgabe 20

Nenne drei verschiedene Anwendungen von Graphen.

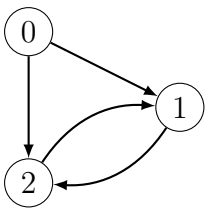
Aufgabe 21

Stelle den gerichteten Graphen (a) als Adjazenzmatrix (b) als Adjazenzliste dar.



Aufgabe 22

Stelle den gerichteten Graphen



in Python irgendwie durch Listen dar.

Aufgabe 23

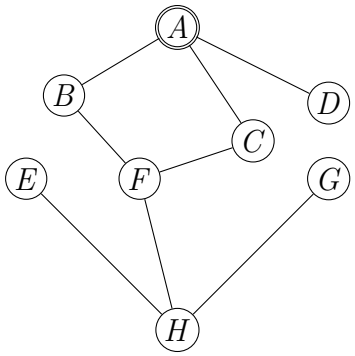
Das folgende Python-Codefragment stellt einen gerichteten Graphen dar.

```
G = [[2, 3], [], [2], [0, 1, 2]]
```

Skizziere diesen Graphen.

Aufgabe 24

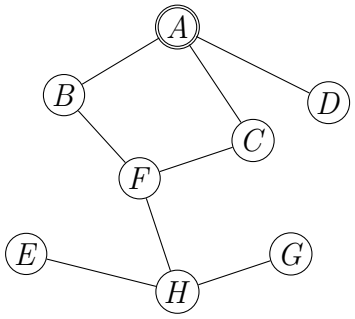
Führe eine Tiefensuche gemäss der rechts angegebenen Adjazenzliste durch.



von	nach
A	C D B
B	F A
C	A F
D	A
E	H
F	C H B
G	H
H	F E G

Aufgabe 25

Führe eine Breitensuche auf dem gegebenen Graphen durch, wobei Nachbarknoten in alphabetischer Reihenfolge besucht werden sollen.



Aufgabe 26

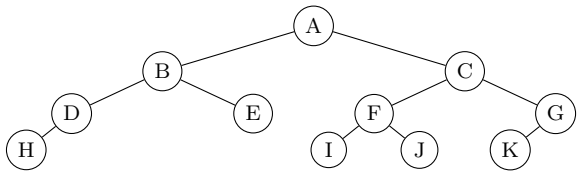
Beschreibe die Datenstruktur *Baum* so genau wie möglich.

Aufgabe 27

Nenne zwei Anwendungen für Bäume als Datenstruktur.

Aufgabe 28

Beantworte die Fragen zum folgenden Baum:



(a) Welchen Schlüssel hat die Wurzel?

(b) Welche Kinder hat der Knoten mit dem Schlüssel C?

(c) Welche Geschwister hat der Knoten mit dem Schlüssel H?

(d) Welchen Elternknoten hat der Knoten mit dem Schlüssel E?

(e) Welches sind die Blätter des Baums?

(f) Welches sind die inneren Knoten des Baums?

(g) Welche Tiefe hat der Knoten mit dem Schlüssel D?

(h) Welche Höhe hat der Baum?

Aufgabe 29

Skizziere den in Zeile 12–16 vom Python-Code erzeugten Baum.

```
1  class Tree:
2
3      def __init__(self, key):
4          self.root = key
5          self.edges = dict()
6
7      def add_children(self, parent, children):
8          self.edges[parent] = children
9          for c in children:
10             self.edges[c] = []
11
12 t = Tree('A')
13 t.add_children('A', ['M', 'K'])
14 t.add_children('M', ['F', 'T'])
15 t.add_children('T', ['S', 'E'])
16 t.add_children('K', ['B', 'U'])
```

Aufgabe 30

Gib die Folge der Knoten an, wenn der folgende Baum in (a) Preorder-Reihenfolge (b) Postorder-Reihenfolge durchlaufen wird.