Hamburg | Behörde für Schule, Familie und Berufsbildung

Schriftliche Abiturprüfung
Schuljahr 2025/2026

Informatik
auf erhöhtem Anforderungsniveau

an allgemeinbildenden gymnasialen Oberstufen

Haupttermin
Mittwoch, 22. April 2026, 09:00 Uhr

Unterlagen für die Lehrkräfte

Diese Unterlagen sind nicht für die Prüflinge bestimmt.

Diese Unterlagen enthalten:

1. Allgemeines[1]

- Weisen Sie bitte die Prüflinge auf die allgemeinen Arbeitshinweise hin.
- Die Arbeitszeit beträgt **300 Minuten**.
- Eine gesonderte Lese- oder Auswahlzeit wird nicht gewährt.
- Hilfsmittel: Taschenrechner (nicht programmierbar, nicht grafikfähig), Rechtschreibwörterbuch

[1] Entsprechend der „Richtlinie über die Gewährung von Erleichterungen für neu zugewanderte Schülerinnen, Schüler und Prüflinge bei Sprachschwierigkeiten in der deutschen Sprache“ (MBlSchul Nr. 08, 7. Oktober 2016, S. 60) werden für die betroffenen Prüflinge die folgenden Erleichterungen gewährt:
- Die Bearbeitungszeit wird um 30 Minuten auf **330 Minuten** erhöht.
- Ein nicht-elektronisches Wörterbuch Deutsch – Herkunftssprache / Herkunftssprache – Deutsch wird bereitgestellt.

2. Rückmeldebogen für die Zweitdurchsicht

Bitte umgehend ausfüllen und

a) an die Schule senden, die die externe Zweitdurchsicht übernimmt,

oder ggf.

b) an die Kollegin/den Kollegen geben, die/der die interne Zweitdurchsicht übernimmt.

Information für die Zweitdurchsicht

Fach:
Informatik
auf erhöhtem Anforderungsniveau Kurs-Nummer: _____

Bearbeitet wurden die folgenden Aufgaben:

Aufgaben-Nr.		Anzahl	
I	von		Prüflingen
II	von		Prüflingen
III	von		Prüflingen

3. Hinweise zu den Aufgaben

- Die Lehrkraft erhält **drei** Aufgaben und reicht diese an die Prüflinge weiter.
- Die Prüflinge wählen **eine** der Aufgaben II oder III aus und bearbeiten diese.
- Sie müssen die Aufgabe I bearbeiten.
- Sie vermerken auf dem Deckblatt und der Reinschrift, welche Aufgaben sie bearbeitet haben.

4. Hinweise zum Korrekturverfahren

- Die Korrekturen werden gemäß der „Richtlinie für die Aufgabenstellung und Bewertung der Leistungen in der Abiturprüfung“ (Abiturrichtlinie) in der geltenden Fassung vorgenommen.
- Die für das Fach zuständige Lehrkraft begutachtet die Arbeiten unter Beachtung zentraler Bewertungsvorgaben und unter Kennzeichnung ihrer Vorzüge und Mängel, der richtigen Lösungen und der Fehler und bewertet jede Arbeit mit einer Punktzahl (siehe Erstkorrekturbogen). Entwürfe können ergänzend zur Bewertung herangezogen werden.
- Jede Arbeit wird sodann von der zweiten Fachlehrkraft durchgesehen, die sich entweder (auf dem Erstkorrekturbogen) der Bewertung durch die für das Fach zuständige Lehrkraft anschließt oder (auf dem Zweitdurchsichtbogen) ein ergänzendes Gutachten mit Bewertung anfertigt.
- Die oder der Vorsitzende des Fachprüfungsausschusses legt die endgültige Punktzahl fest. Beträgt die Differenz der im Erstgutachten und im ergänzenden Gutachten erteilten Punktzahlen nicht mehr als drei Punkte, bildet sie oder er den Mittelwert beider Punktzahlen; eine gebrochene Zahl wird zur nächsten vollen Punktzahl aufgerundet. Beträgt die Differenz der im Erstgutachten und im ergänzenden Gutachten erteilten Punktzahlen mehr als drei Punkte, legt die oder der Vorsitzende des Fachprüfungsausschusses die endgültige Punktzahl in Auseinandersetzung mit den erstellten Gutachten entsprechend dem Erfordernis der Einheitlichkeit und Vergleichbarkeit der Bewertung der Prüfungsleistungen fest und begründet die Entscheidung auf dem Formular „Festlegung der Note bei einer Differenz der im Erstgutachten und im ergänzenden Gutachten erteilten Punktzahlen von mehr als drei Notenpunkten“.
- Die Bewertungsbögen sind als Download zu finden auf der Plattform Prüfungen (vormals HERA) in LMS.

5. Bewertung und Erwartungshorizont

I. Bewertung

Jeder Aufgabe sind 50 Bewertungseinheiten (BE) zugeordnet. In allen Teilaufgaben werden nur ganze BE vergeben. Insgesamt sind 100 BE erreichbar. Bei der Festlegung von Notenpunkten gilt die folgende Tabelle:

Erbrachte Leistung (in BE)	Notenpunkte
≥ 95	15
≥ 90	14
≥ 85	13
≥ 80	12
≥ 75	11
≥ 70	10
≥ 65	9
≥ 60	8

Erbrachte Leistung (in BE)	Notenpunkte
≥ 55	7
≥ 50	6
≥ 45	5
≥ 40	4
≥ 33	3
≥ 27	2
≥ 20	1
< 20	0

Für die Erteilung der **Note gut** (11 Punkte) ist mindestens erforderlich, dass die Prüflinge 75 % der erwarteten Gesamtleistung erbracht haben. Dabei muss die Prüfungsleistung in ihrer Gliederung, in der Gedankenführung, in der Anwendung fachmethodischer Verfahren sowie in der fachsprachlichen Artikulation den Anforderungen voll entsprechen.

Für die Erteilung der **Note ausreichend** (5 Punkte) ist mindestens erforderlich, dass die Prüflinge mindestens 45 % der erwarteten Gesamtleistung und über den Anforderungsbereich I hinaus Leistungen in einem weiteren Anforderungsbereich erbracht haben.

Die zwei voneinander unabhängigen Aufgaben der Prüfungsaufgabe werden jeweils mit 50 Bewertungseinheiten bewertet. Die erbrachte Gesamtleistung ergibt sich aus der Summe der Bewertungseinheiten in den beiden Aufgaben.

Schwerwiegende und gehäufte Verstöße gegen die sprachliche Richtigkeit oder gegen die äußere Form führen zu einem Abzug von bis zu zwei Punkten in einfacher Wertung. Mangelhafte Gliederung, Fehler in der Fachsprache, Ungenauigkeiten in Zeichnungen oder unzureichende oder falsche Bezüge zwischen Zeichnungen und Text sind als fachliche Fehler zu werten. Ein Abzug für Verstöße gegen die sprachliche Richtigkeit soll nicht erfolgen, wenn diese bereits Gegenstand der fachspezifischen Bewertungsvorgaben sind.

II. Erwartungshorizont

Bei den auf den folgenden Seiten dargestellten erwarteten Schülerleistungen handelt es sich um Lösungsskizzen. Oft sind aber Lösungsvarianten möglich, die in der Skizze nur zum Teil beschrieben werden konnten. Grundsätzlich gilt deshalb, dass alle Varianten, die zu richtigen Lösungen führen, mit voller Punktzahl bewertet werden, unabhängig davon, ob die gewählte Variante in der Lösungsskizze aufgeführt ist oder nicht.

Kursiv gedruckte Passagen sind Hinweise an die korrigierenden Lehrkräfte. Sie sind nicht Bestandteile der erwarteten Schülerleistung.

Aufgabe I: Schiffskarten (Python-Version)

Schwerpunkt: Objektorientierte Programmierung und Modellierung

	Lösungsskizze	Zuordnung Bewertung		
		I	II	III
a)	Es müssen Objekte der Klassen `Schiff`, `Hafen` und `AISStation` erzeugt werden.		3	
	In der Objektorientierten Programmierung ist eine Klasse die grundlegende Struktur für die Definition von Objekten. Sie dient als Bauplan, um Objekte zu erstellen, die bestimmte Eigenschaften (Attribute) und Verhaltensweisen (Methoden) haben. In diesem Projekt werden beispielsweise Objekte für konkrete Schiffe erzeugt, die gleiche Attribute und Methoden haben, aber unterschiedliche Attributwerte wie Position und Name.	3	3	
b)	Es gibt in dem Modell Nutzungsbeziehungen (*hat-ein-Beziehung*) zwischen Elementverwaltung und den Elementen und von den Elementen zur Zeichenfläche. Zudem werden einige Klassen von Oberklassen abgeleitet (*Vererbung, Generalisierung/Spezialisierung, ist-ein-Beziehung*): Ein allgemeiner Typ Element für alles, was auf der Karte angezeigt wird, ist Grundlage für die Klassen `Schiff`, `Hafen` und `AISStation`.	3	3	
	Die Vererbungsbeziehung erkennt man im Kopf der Klassendefinition an der Angabe der Oberklasse in Klammern hinter dem Klassennamen, konkret also beispielsweise `class Schiff(Element):` Die Verwendungsbeziehung erkennt man im Programmtext an einem Attribut der verwendenden Klasse oder einer Zugriffsmethode auf ein Objekt der verwendeten Klasse.	4		
	In einer abstrakten Klasse werden die gemeinsamen Eigenschaften anderer Klassen ausmodelliert, ohne dass es Instanzen dieser Klasse geben kann bzw. soll. Man kann also eine Oberklasse implementieren und verhindern, dass ein Objekt von einem allgemeinen Typ `Element` erzeugt wird, für das es bspw. keine Form der Darstellung geben kann. In der abstrakten Klasse kann aber erzwungen werden, dass in allen Unterklassen eine Methode implementiert ist, die für die Darstellung sorgt.	3	2	
c)	In einer Unterklasse kann für den konkreten Schiffstyp die Farbe der Darstellung einheitlich festgelegt werden, indem sie im Konstruktor gesetzt wird. Ebenso kann aber in einer allgemeinen Klasse `Schiff` die Farbe vom Wert des Attributs `schiffstyp` abgeleitet werden.		2	
	Nach dem Anlegen müsste das Objekt, das ein Schiff repräsentiert, entfernt werden und ein neues Objekt mit identischen Objektwerten, aber von einem anderen Typ erzeugt werden. Ebenso müsste umgekehrt verfahren werden, wenn ein Schiff wieder ausläuft. Das widerspricht der Idee der objektorientierten Modellierung, nach der ein Objekt über die Lebensdauer seinen Zustand, nicht aber seinen Typ ändert.		2	1

	Polymorphie ermöglicht, dass ein Bezeichner abhängig von seiner Verwendung Objekte unterschiedlichen Datentyps annimmt. Dadurch kann der Aufruf einer Methode gleicher Signatur zu unterschiedlichen Ergebnissen führen. In diesem Beispiel würde also das Symbol je nach Schiffstyp eine andere Farbe erhalten.		2	
d)	Mit dem Code wird eine Liste mit dem Namen `schiffsliste` erzeugt. Eine Liste ist hierfür gut geeignet. Die Zahl der verwalteten Schiffe kann sich ändern, so dass eine flexible Datenstruktur verwendet werden sollte. Es ist aber nicht sinnvoll, als Datenstruktur ein Tupel zu nutzen, da die Zahl der Schiffe auf der Karte variabel ist.		2	1
e)	Hier sind verschiedene Lösungen denkbar, etwa • Auswahl anhand der eindeutigen MMSI-Nummer oder • Übergabe einer Objektreferenz und Prüfung auf Objektgleichheit mit *is*. Der Prüfling soll insbesondere den Zugriff auf alle Elemente in der Liste, bis das gesuchte Schiff gefunden wurde oder das Ende der Liste erreicht wurde, korrekt implementieren. *Je nach unterrichtlichen Voraussetzungen muss nicht zur Abwertung führen,* *• dass `remove` nicht mit einer `for-each-Schleife` kombiniert werden darf, wenn man nicht auf einer Kopie der Liste arbeitet, oder* *• wenn der Befehl `return` oder `break` vergessen wird, was zusammen mit `remove` zu einem Problem führen kann.* Beispiel: `def entferne_schiff(mmsi):` `  for i in range(len(schiffsliste)):` `  akt = schiffsliste[i]` `  if akt.gibMMSI() == mmsi:` `  schiffsliste.remove[akt]` `  break`		2	2
	Die gewählte Lösung muss nachvollziehbar erläutert werden.		2	
f)	Notwendig sind Attribute für die Länge und die Breite des Schiffs insgesamt. Zudem muss ein Punkt als Ursprung der Form definiert werden, von dem aus die Position der GPS-Antenne gerechnet wird. *Das kann bspw. die Spitze am Bug sein oder die Backbord-Ecke des Hecks.* Hierfür sind zwei weitere Attribute für die Abstände in den beiden Dimensionen notwendig.		2	2
	Die Position des Kreises darf nicht absolut von einem Punkt aus festgelegt werden, sondern muss relativ zur Größe der Darstellung errechnet werden.			2
g)	`AIS-Gebiet` ist als eigenständige Klasse zu modellieren, die eine Nutzungsbeziehung zu `AIS-Station` hat. *Eine Darstellung im Diagramm ist nicht gefordert.*			2
	Der Methodenaufruf an ein `AIS-Gebiet` führt zu Anfragen an die dort jeweils verwalteten `AIS-Stationen`. Deren Listen bekannter Schiffe müssen dann, bspw. anhand der MMSI, zu einer Liste zusammengeführt werden, die den Rückgabewert des ursprünglichen Methodenaufrufs bildet.			2
	Insgesamt 50 BE	13	25	12

Aufgabe I: Schiffskarten (Java-Version)

Schwerpunkt: Objektorientierte Programmierung und Modellierung

	Lösungsskizze	Zuordnung Bewertung		
		I	II	III
a)	Es müssen Objekte der Klassen `Schiff`, `Hafen` und `AISStation` erzeugt werden.		3	
	In der Objektorientierten Programmierung ist eine Klasse die grundlegende Struktur für die Definition von Objekten. Sie dient als Bauplan, um Objekte zu erstellen, die bestimmte Eigenschaften (Attribute) und Verhaltensweisen (Methoden) haben. In diesem Projekt werden beispielsweise Objekte für konkrete Schiffe erzeugt, die gleiche Attribute und Methoden haben, aber unterschiedliche Attributwerte wie Position und Name.	3	3	
b)	Es gibt in dem Modell Nutzungsbeziehungen (*hat-ein-Beziehung*) zwischen Elementverwaltung und den Elementen und von den Elementen zur Zeichenfläche. Zudem werden einige Klassen von Oberklassen abgeleitet (*Vererbung, Generalisierung/Spezialisierung, ist-ein-Beziehung*): Ein allgemeiner Typ Element für alles, was auf der Karte angezeigt wird, ist Grundlage für die Klassen Schiff, Hafen und AISStation.	3	3	
	Die Vererbungsbeziehung erkennt man im Kopf der Klassendefinition an der Angabe `extends <Oberklasse>`, konkret also beispielsweise `public class Schiff extends Element;` Die Verwendungsbeziehung erkennt man im Programmtext an einem Attribut der verwendenden Klasse oder einer Zugriffsmethode auf ein Objekt der verwendeten Klasse.	4		
	In einer abstrakten Klasse werden die gemeinsamen Eigenschaften anderer Klassen ausmodelliert, ohne dass es Instanzen dieser Klasse geben kann bzw. soll. Man kann also eine Oberklasse implementieren und verhindern, dass ein Objekt von einem allgemeinen Typ `Element` erzeugt wird, für das es bspw. keine Form der Darstellung geben kann. In der abstrakten Klasse kann aber erzwungen werden, dass in allen Unterklassen eine Methode implementiert ist, die für die Darstellung sorgt.	3	2	
c)	In einer Unterklasse kann für den konkreten Schiffstyp die Farbe der Darstellung einheitlich festgelegt werden, indem sie im Konstruktor gesetzt wird. Ebenso kann aber in einer allgemeinen Klasse `Schiff` die Farbe vom Wert des Attributs `schiffstyp` abgeleitet werden.		2	
	Nach dem Anlegen müsste das Objekt, das ein Schiff repräsentiert, entfernt werden und ein neues Objekt mit identischen Objektwerten, aber von einem anderen Typ erzeugt werden. Ebenso müsste umgekehrt verfahren werden, wenn ein Schiff wieder ausläuft. Das widerspricht der Idee der objektorientierten Modellierung, nach der ein Objekt über die Lebensdauer seinen Zustand, nicht aber seinen Typ ändert.		2	1
	Polymorphie ermöglicht, dass ein Bezeichner abhängig von seiner Verwendung Objekte unterschiedlichen Datentyps annimmt. Dadurch		2	

	kann der Aufruf einer Methode gleicher Signatur zu unterschiedlichen Ergebnissen führen. In diesem Beispiel würde also das Symbol je nach Schiffstyp eine andere Farbe erhalten.			
d)	Grundsätzlich stellt dieses Vorgehen eine mögliche Lösung der Anforderung dar, alle Schiffe in einer Datenstruktur zu halten. Es ist aber nicht sinnvoll, die Datenstruktur Array zu nutzen, da die Zahl der Schiffe auf der Karte variabel ist.		2	1
e)	Hier sind verschiedene Lösungen denkbar, etwa • Auswahl anhand der eindeutigen MMSI-Nummer oder • Übergabe einer Objektreferenz und Prüfung auf Objektgleichheit mit *equals*. Der Prüfling soll insbesondere den Zugriff auf alle Elemente in der ArrayList, bis das gesuchte Schiff gefunden wurde oder das Ende der Liste erreicht wurde, korrekt implementieren. *Je nach unterrichtlichen Voraussetzungen muss nicht zur Abwertung führen,* *• dass `remove` nicht mit einer `for-each-Schleife` kombiniert werden darf, oder* *• wenn der Befehl `break;` oder `return;` vergessen wird, was zusammen mit remove zu einem Problem führen kann, wenn man keinen Iterator verwendet.* Beispiel: `public void entferneSchiff(int mmsi)` `{` `Schiff akt;` `for (int i = 0; i < schiffsliste.size(); i++)` `{` `akt = schiffliste.get(i);` `if (akt.gibMMSI() == mmsi)` `{` `schiffsliste.remove(i);` `break;` `}` `}` `}`		2	2
	Die gewählte Lösung muss nachvollziehbar erläutert werden.		2	
f)	Notwendig sind Attribute für die Länge und die Breite des Schiffs insgesamt. Zudem muss ein Punkt als Ursprung der Form definiert werden, von dem aus die Position der GPS-Antenne gerechnet wird. *Das kann bspw. die Spitze am Bug sein oder die Backbord-Ecke des Hecks.* Hierfür sind zwei weitere Attribute für die Abstände in den beiden Dimensionen notwendig.		2	2
	Die Position des Kreises darf nicht absolut von einem Punkt aus festgelegt werden, sondern muss relativ zur Größe der Darstellung errechnet werden.			2

g)	`AIS-Gebiet` ist als eigenständige Klasse zu modellieren, die eine Nutzungsbeziehung zu `AIS-Station` hat. *Eine Darstellung im Diagramm ist nicht gefordert.*			2
	Der Methodenaufruf an ein `AIS-Gebiet` führt zu Anfragen an die dort jeweils verwalteten `AIS-Stationen`. Deren Listen bekannter Schiffe müssen dann, bspw. anhand der MMSI, zu einer Liste zusammengeführt werden, die den Rückgabewert des ursprünglichen Methodenaufrufs bildet.			2
	Insgesamt 50 BE	13	25	12

Aufgabe II: Bitcoin (Scheme-Version)

Schwerpunkt: Datensicherheit in verteilten Systemen

	Lösungsskizze	Zuordnung Bewertung		
		I	II	III
a)	Es wird dadurch ausschließlich die Integrität der Daten bestätigt.	6		
	Der Hash ist öffentlich einsehbar, genauso wie die Transaktionsdaten. Damit ist Vertraulichkeit und Authentizität nicht gefragt. Wichtig ist lediglich die Integrität der Daten, die auch im Nachhinein sichergestellt sein muss. Da der Hash eines Vorgänger-Blocks jeweils in den nachfolgenden Block mit eingeht, ist es nachträglich nicht möglich, einen vorhergehenden Block zu ändern, ohne dass es in nachfolgenden Blöcken auffallen würde. *Anmerkung: Mit dem Block-Hash wird lediglich der Header eines Blockes verifiziert, die einzelnen Transaktionen in dem Block, also die Transaktionsdaten, werden nochmal separat gehasht und der Hash dann in den Blockhash integriert – diese Unterscheidung wird jedoch von den SuS nicht erwartet, um diese Aufgabe korrekt zu beantworten.*	2	3	
b)	Durch die `(modulo … 16777216)` Rechnung wird jeder resultierende Hash auf die max. Zahl 16777215 begrenzt. Damit ist die max. Länge des Hashes vorgegeben.	2	3	
	Die `hashHilf` Funktion nimmt eine Liste von `chars` entgegen (aus denen sich die zu hashende Nachricht zusammensetzt). Dazu wird ein fortlaufender `index` übergeben, der bei jedem Rekursionsschritt um 1 hochgezählt wird. Ebenso bekommt die Funktion einen Wert `multi` übergeben, dieser Wert wird in jedem rekursiven Schritt mit dem Wert `faktor` multipliziert (zu Beginn ist `multi` auf 1 gesetzt, wird aber in jedem rekursiven Schritt mit dem Wert `faktor` multipliziert, `faktor` selbst bleibt konstant). Als Parameter erhält die Funktion auch den Parameter `summe`, der in jedem rekursiven Schritt neu aufaddiert wird und am Ende den Wert liefert, der zurück an den Aufrufer geht. Da in den rekursiven Aufrufen die Werte für `summe` und `multi` bei längeren Texten zu groß werden können, muss zusätzlich der Modulo Wert *MOD* mit übergeben werden, um diesen in jedem Rekursionsschritt auch auf die Zwischenwerte anzuwenden. Die Funktion wird solange rekursiv aufgerufen, bis die Liste aus `chars` abgearbeitet ist. In jedem Schritt wird die Summe gebildet aus dem vorhergehenden Summen-Wert plus einem Wert, in den der aktuelle `chars` (als Ascii-Wert) eingeht und die übergebenen Parameter `index` und `multi` (die sich in jedem rekursiven Schritt ändern). Der am Ende errechnete Wert `summe` wird nochmal über die Modulo-Rechnung auf einen Wert reduziert, der auf jeden Fall kleiner als die Zahl 16777216 ist. Somit behält die Ausgabe der Funktion `einfacherHash` immer eine maximale Länge, was die Grundlage für einen Hash ist.	2	3	3

c)	Ähnliche Daten sollten keine ähnlichen Hashes erzeugen, weil man sich dem Ergebnis sonst Schritt für Schritt nähern könnte. So aber ist es ein zufälliger Prozess und man muss alle Möglichkeiten durchprobieren, bis man zufällig erfolgreich ist.		2	3
d)	Die Funktion `mineBlock` berechnet einen Hash in Hexadezimal-Darstellung aus: den gegebenen Daten `daten`, dem vorhergehenden Hash `prevHash` und einem Zahlwert `nonce`, was alles zusammen in einen String gepackt wird – wozu dann anschließend der Hash berechnet wird. In der `minBlockHilf` Funktion wird der Wert `nonce` in jedem Schritt um 1 hochgezählt und entsprechend zu den Daten hinzugefügt. Da nicht irgendein beliebiger hexadezimaler Hash berechnet werden soll, wird als Parameter eine Zahl `schwierigkeit` an die `mineBlockHilf` Funktion übergeben, die angibt, wieviel führende Nullen der hexadezimale Hash am Ende der Berechnung haben muss. Die führenden Nullen entstehen bei der Umrechnung des dezimalen Hashes in die Hexadezimal-Darstellung, da die Funktion `fuelleString` den hexadezimalen String nach der Umrechnung über die Funktion `int->hexString` vorne mit Nullen auffüllt, wenn der errechnete hexString kürzer ist als die maximale Länge, die in mineBlock über den Parameter `maxStellenHash` angegeben wird. Da die `einfacherHash` Funktion keine kontinuierlichen Ergebnisse liefert, wenn man den Zahlwert `nonce` in jedem Schritt um 1 erhöht, muss in `mineBlock` zu jedem neuen `nonce` Wert der neue hexadezimale Hash errechnet werden und es muss abgeglichen werden, ob dieser die vorgegebene Schwierigkeit erfüllt (was durch die Funktion `hashTrifftSchwierigkeit?` bzw. durch `checkPrefix?` erreicht wird). Sobald die Funktion `hashTrifftSchwierigkeit?` #t liefert, ist die Funktion `mineBlockHilf` abgeschlossen und es werden der `nonce` Wert, der errechnete finale `hashWert` und der dazu passende hexadezimale Hash als Liste ausgegeben.		6	3
	`(define (stringReferenz hexString position)` `(stringRefHilf (string->list hexString) position 0))` `(define (stringRefHilf chars position zaehler)` `(cond ((null? chars) #\?)` `((equal? position zaehler) (first chars))` `(else (stringRefHilf (rest chars)` `position` `(+ zaehler 1)))))`		5	3
e)	„Proof-of-Work-Prozess" ist eine sinnvolle Bezeichnung für den Vorgang des Minens, weil man einen passenden Hash nicht über einen bestimmten Algorithmus finden kann, sondern ihn nur über Ausprobieren und Ausdauer mit verschiedenen Nonces findet – dieser Vorgang verbraucht i.d.R. viel Rechenzeit und ist daher arbeitsintensiv (für den Rechner).		3	1
	Insgesamt 50 BE	12	25	13

Aufgabe II: Bitcoin (Haskell-Version)

Schwerpunkt: Datensicherheit in verteilten Systemen

	Lösungsskizze	Zuordnung Bewertung		
		I	II	III
a)	Es wird dadurch ausschließlich die Integrität der Daten bestätigt.	6		
	Der Hash ist öffentlich einsehbar, genauso wie die Transaktionsdaten. Damit ist Vertraulichkeit und Authentizität nicht gefragt. Wichtig ist lediglich die Integrität der Daten, die auch im Nachhinein sichergestellt sein muss. Da der Hash eines Vorgänger-Blocks jeweils in den nachfolgenden Block mit eingeht, ist es nachträglich nicht möglich, einen vorhergehenden Block zu ändern, ohne dass es in nachfolgenden Blöcken auffallen würde. *Anmerkung: Mit dem Block-Hash wird lediglich der Header eines Blocks verifiziert, die einzelnen Transaktionen in dem Block, also die Transaktionsdaten, werden nochmal separat gehasht und der Hash dann in den Blockhash integriert – diese Unterscheidung wird jedoch von den SuS nicht erwartet, um diese Aufgabe korrekt zu beantworten.*	2	3	
b)	Durch die Rechnung `modulo 16777216` wird jeder resultierende Hash auf die max. Zahl 16777215 begrenzt. Damit ist die max. Länge des Hashes vorgegeben.	2	3	
	Die Funktion `hashHilf` nimmt eine Liste `chars` von Buchstaben entgegen (aus denen sich die zu hashende Nachricht zusammensetzt). Dazu wird ein fortlaufender `index` übergeben, der bei jedem Rekursionsschritt um 1 hochgezählt wird. Ebenso bekommt die Funktion einen Wert `multi` übergeben, dieser Wert wird in jedem rekursiven Schritt mit dem Wert `faktor` multipliziert. Zu Beginn ist `multi` auf 1 gesetzt, wird aber in jedem rekursiven Schritt mit dem Wert `faktor` multipliziert; `faktor` selbst bleibt konstant. Als Parameter erhält die Funktion auch den Parameter `summe`, der in jedem rekursiven Schritt neu aufaddiert wird und am Ende den Wert liefert, der zurück an den Aufrufer geht. Da in den rekursiven Aufrufen die Werte für `summe` und `multi` bei längeren Texten zu groß werden können, muss zusätzlich der Modulo Wert *MOD* mit übergeben werden, um diesen in jedem Rekursionsschritt auch auf die Zwischenwerte anzuwenden. Die Funktion wird solange rekursiv aufgerufen, bis die Liste aus `chars` abgearbeitet ist. In jedem Schritt wird die Summe gebildet aus dem vorhergehenden Summen-Wert plus einem Wert, in den der aktuelle Buchstabe aus `chars` (als ASCII-Wert) eingeht und die übergebenen Parameter `index` und `multi`, die sich in jedem rekursiven Schritt ändern. Der am Ende errechnete Wert `summe` wird noch einmal über die Modulo-Rechnung auf einen Wert reduziert, der auf jeden Fall kleiner als die Zahl 16777216 ist. Somit behält die Ausgabe der Funktion `einfacherHash` immer eine maximale Länge, was die Grundlage für einen Hash ist.	2	3	3

c)	Ähnliche Daten sollten keine ähnlichen Hashes erzeugen, weil man sich dem Ergebnis sonst Schritt für Schritt nähern könnte. So aber ist es ein zufälliger Prozess und man muss alle Möglichkeiten durchprobieren, bis man erfolgreich ist.		2	3
d)	Die Funktion `mineBlock` berechnet einen Hash in Hexadezimal-Darstellung aus den gegebenen Daten `daten`, dem vorhergehenden Hash `prevHash` und einem Zahlwert `nonce`, was alles zusammen in einem String zusammengefasst wird, zu dem dann anschließend der Hash berechnet wird. In der Funktion `minBlockHilf` wird der Wert `nonce` in jedem Schritt um 1 hochgezählt und entsprechend zu den Daten hinzugefügt. Da nicht irgendein beliebiger hexadezimaler Hash berechnet werden soll, wird als Parameter eine Zahl `schwierigkeit` an die `mineBlockHilf` Funktion übergeben, die angibt, wieviel führende Nullen der hexadezimale Hash am Ende der Berechnung haben muss. Die führenden Nullen entstehen bei der Umrechnung des dezimalen Hashes in die Hexadezimal-Darstellung, da die Funktion `fuelleString` den hexadezimalen String nach der Umrechnung über die Funktion `intToHexString` vorne mit Nullen auffüllt, wenn der errechnete hexString kürzer ist als die maximale Länge, die in mineBlock über den Parameter `maxStellenHash` angegeben wird. Es muss abgeglichen werden, ob dieser die vorgegebene Schwierigkeit erfüllt, was durch die Funktion `hashTrifftSchwierigkeit` bzw. durch `checkPrefix` erreicht wird. Sobald die Funktion `hashTrifftSchwierigkeit` True liefert, ist die Auswertung der Funktion `mineBlockHilf` abgeschlossen, und es werden der `nonce` Wert, der errechnete finale `hashWert` und der dazu passende hexadezimale Hash als Liste ausgegeben.		6	3
	`stringReferenz hexString position` `  = stringRefHilf hexString position 0` `-- stringRefHilf chars position zaehler` `stringRefHilf []  _  = '?'` `stringRefHilf (c:cs) position zaehler` `  | position == zaehler =  c` `  | otherwise  =  stringRefHilf cs position` `  (zaehler + 1)`		5	3
e)	„Proof-of-Work-Prozess" ist eine sinnvolle Bezeichnung für den Vorgang des Minens, weil man einen passenden Hash nicht über einen bestimmten Algorithmus finden kann, sondern ihn nur über Ausprobieren und Ausdauer mit verschiedenen Nonces findet – dieser Vorgang verbraucht i. d. R. viel Rechenzeit und ist daher arbeitsintensiv für den Rechner.		3	1
	Insgesamt 50 BE	12	25	13

Aufgabe III: MENACE (Scheme-Version)

Schwerpunkt: Intelligente Suchverfahren

	Lösungsskizze	Zuordnung Bewertung		
		I	II	III
a)	Die SuS sollen beschreiben, dass mit einer tiefen Liste gearbeitet wird. (*Sie brauchen diesen Fachausdruck nicht zu nennen, die Schachtelung soll aber im Text erkennbar sein.*) Sie sollen beschreiben, dass am Muster erkennbar ist, dass die Teillisten für die Zeilen der Darstellung stehen. Die SuS können angeben, dass auch eine Version mit einer flachen Liste möglich ist. Sie sollen erläutern, dass auf flache Listen zwar einfacher zugegriffen werden kann, dass tiefe Listen aber besser die Problemstruktur beschreiben und dadurch die Art der Zugriffe verständlicher ist.	4	1 2	 1
b)	Im ersten Schritt werden alle Spielstände erzeugt, die für den ersten Kreis möglich sind. Das sind neun Möglichkeiten. Zu jedem dieser Zustände werden danach die acht möglichen Positionen mit dem ersten Kreuz auf einem der freien Felder bestimmt, insgesamt also 72 Varianten. Zu denen werden jeweils die sieben freien Felder mit dem Kreis belegt. Dadurch entstehen 504 Varianten. Hier sind zwei Aspekte zu berücksichtigen: Einmal unterscheiden sich nicht die beiden Spielsituationen, bei denen die beiden Kreise zwar in unterschiedlicher Reihenfolge, allerdings auf die selben Felder gesetzt worden sind. Weiterhin sollte erkannt werden, dass die Drehung des Felds um 90°, 180° und 270° genau so wenig zu einer echten neuen Spielsituation führt wie die Spiegelung an einer Diagonalen.	3	2 2	 3

c)	Eine eindeutige Beschreibung der Vorgehensweise der Funktionen ist auch zulässig. Ein möglicher Funktionstext: `(define` `  (hatGewonnenZeile spielstand)` `  (cond` `  ((null? spielstand)` `  #f)` `  ((dreiGleich (first spielstand))` `  #t)` `  (else` `  (hatGewonnenZeile (rest spielstand)))` `  )` `  )` `(define` `  (dreiGleich drei)` `  (cond` `  ((equal? (first drei) " ")` `  #f)` `  ((not (equal? (first drei) (second drei)))` `  #f)` `  ((not (equal? (second drei) (third drei)))` `  #f)` `  (else` `  #t)` `  )` `  )` Eine Lösung ohne Rekursion durch Verknüpfung der Bedingungen mit **and** ist ebenfalls sinnvoll.		3	2
d)	Die Funktion füllt das Feld abwechselnd mit den beiden Zeichen **"o"** und **"x"**. Angabe einer Rückgabe: **(("o" "x" "o") ("x" "o" "x") ("o" "x" "o"))** Eine textliche Beschreibung ist auch zulässig. Für eine Tiefensuche fehlt das Generieren und Untersuchen von Alternativen in jedem Schritt. Ein möglicher Programmtext: `(define` `  (neuesZeichen zeichen)` `  (cond` `  ((equal? zeichen "x")` `  "o")` `  (else` `  "x")` `  )` `)	3	2 2 2	1 1 1
e)	Es bieten sich zwei grundsätzlich verschiedene Vorgehensweisen an. (*Es reicht, wenn eine Vorgehensweise sinnvoll beschrieben wird.*) • Bei einer Vorgehensweise wird zunächst geprüft, ob es überhaupt noch ein freies Feld gibt. Wenn das nicht der Fall ist, hat kein Mitspieler gewonnen. Anderenfalls wird aus allen Feldern des Spielstands eines zufällig ausgesucht und geprüft, ob das Feld noch unbesetzt ist. Dieser Schritt wird solange wiederholt, bis ein freies Feld gefunden worden ist, das dann mit dem Zeichen besetzt werden kann. • Bei der anderen Vorgehensweise werden zunächst alle noch freien Felder ermittelt und, wenn es welche gibt, aus ihnen eines zufällig ausgewählt.	3	2	

<table>
<tr><td></td><td>Die weiteren Schritte bleiben in beiden Fällen dieselben.

Bei einer realen Spielsituation wählen die Spieler in der Regel nicht zufällig, sondern nach eigenen Kriterien. Dadurch wird es in der Regel zu anderen bevorzugten Auswahlen und damit zu anderen Zügen kommen. Auch durch das zufällige Auswählen kann das Programm eine sinnvolle Bewertung der auftretenden Spielstände ausführen, was die Möglichkeit bietet, eher erfolgreiche weitere Schritte von schlechteren zu unterscheiden.</td><td></td><td>4</td><td>1</td></tr>
<tr><td>f)</td><td><pre>(define
    (erhoeheEinen bewertete stand)
    (cond
      ((null? bewertete) '())
      ((equal? (first (first bewertete)) stand)
       (cons
         (list
           (first (first bewertete))
           (+ 1 (second (first bewertete))))
         (rest bewertete)))
       (else
         (cons
           (first bewertete)
           (erhoeheEinen (rest bewertete) stand)))
)
)</pre>
Es ist auch sinnvoll, eine Hilfsfunktion <code>erhoeheBewertung</code> einzusetzen:
<pre>(define
  (erhoeheEinen bewertete stand)
  (cond
    ((null? bewertete) '())
    ((equal? (first (first bewertete)) stand)
     (cons
      (erhoeheBewertung (first bewertete))
      (rest bewertete)))
    (else
     (cons
      (first bewertete)
      (erhoeheEinen (rest bewertete) stand)))
)
)

(define
  (erhoeheBewertung bewerteter)
  (list
   (first bewerteter)
   (+ 1 (second bewerteter))
)
)</pre></td><td></td><td>3</td><td>2</td></tr>
<tr><td colspan="2">Insgesamt 50 BE</td><td>13</td><td>25</td><td>12</td></tr>
</table>

Aufgabe III: MENACE (Haskell-Version)

Schwerpunkt: Intelligente Suchverfahren

	Lösungsskizze	Zuordnung Bewertung		
		I	II	III
a)	Die SuS sollen beschreiben, dass mit einer tiefen Liste gearbeitet wird. (*Sie brauchen diesen Fachausdruck nicht zu nennen, die Schachtelung soll aber im Text erkennbar sein.*) Sie sollen beschreiben, dass am Muster erkennbar ist, dass die Teillisten für die Zeilen der Darstellung stehen.	4	1	
	Die SuS können angeben, dass auch eine Version mit einer flachen Liste möglich ist. Sie sollen erläutern, dass auf flache Listen zwar einfacher zugegriffen werden kann, dass tiefe Listen aber besser die Problemstruktur beschreiben und dadurch die Art der Zugriffe verständlicher ist.		2	1
b)	Im ersten Schritt werden alle Spielstände erzeugt, die für den ersten Kreis möglich sind. Das sind neun Möglichkeiten. Zu jedem dieser Zustände werden danach die acht möglichen Positionen mit dem ersten Kreuz auf einem der freien Felder bestimmt, insgesamt also 72 Varianten. Zu denen werden jeweils die sieben freien Felder mit dem Kreis belegt. Dadurch entstehen 504 Varianten.	3	2	
	Hier sind zwei Aspekte zu berücksichtigen: Einmal unterscheiden sich nicht die beiden Spielsituationen, bei denen die beiden Kreise zwar in unterschiedlicher Reihenfolge, allerdings auf die selben Felder gesetzt worden sind. Weiterhin sollte erkannt werden, dass die Drehung des Felds um 90°, 180° und 270° genau so wenig zu einer echten neuen Spielsituation führt wie die Spiegelung an einer Diagonalen.		2	3
c)	Eine eindeutige Beschreibung der Vorgehensweise der Funktionen ist auch zulässig. Ein möglicher Funktionstext: `(-- hatGewonnenZeile spielstand` `hatGewonnen []  = False` `hatGewonnen (z:rest)` `  | dreiGleich z  = True` `  | otherwise  = hatGewonnenZeile rest` `dreiGleich (a,b,c)` `  | a == ' '  = False` `  | not (a == b)  = False` `  | not (b == c)  = False` `  | otherwise  = True` Eine Lösung ohne Rekursion durch Verknüpfung der Bedingungen mit `and` ist ebenfalls sinnvoll.		3	2

d)	Die Funktion füllt das Feld abwechselnd mit den beiden Zeichen `"o"` und `"x"`. Angabe einer Rückgabe: `[('o','x','o'),('x','o','x'),('o','x','o')]` Eine textliche Beschreibung ist auch zulässig. Für eine Tiefensuche fehlt das Generieren und Untersuchen von Alternativen in jedem Schritt. Ein möglicher Programmtext: `neuesZeichen zeichen` `  | zeichen == 'x'  = 'o'` `  | otherwise  = 'x'`	3	2 2 2	1 1 1
e)	Es bieten sich zwei grundsätzlich verschiedene Vorgehensweisen an. (*Es reicht, wenn eine Vorgehensweise sinnvoll beschrieben wird.*) • Bei einer Vorgehensweise wird zunächst geprüft, ob es überhaupt noch ein freies Feld gibt. Wenn das nicht der Fall ist, hat kein Mitspieler gewonnen. Anderenfalls wird aus allen Feldern des Spielstands eines zufällig ausgesucht und geprüft, ob das Feld noch unbesetzt ist. Dieser Schritt wird solange wiederholt, bis ein freies Feld gefunden worden ist, das dann mit dem Zeichen besetzt werden kann. • Bei der anderen Vorgehensweise werden zunächst alle noch freien Felder ermittelt und, wenn es welche gibt, aus ihnen eines zufällig ausgewählt. Die weiteren Schritte bleiben in beiden Fällen dieselben. Bei einer realen Spielsituation wählen die Spieler in der Regel nicht zufällig, sondern nach eigenen Kriterien. Dadurch wird es in der Regel zu anderen bevorzugten Auswahlen und damit zu anderen Zügen kommen. Auch durch das zufällige Auswählen kann das Programm eine sinnvolle Bewertung der auftretenden Spielstände ausführen, was die Möglichkeit bietet, eher erfolgreiche weitere Schritte von schlechteren zu unterscheiden.	3	2 4	 1
f)	`-- erhoeheEinen bewertete stand` `erhoeheEinen [] _  = []` `erhoeheEinen ((x:y:rest) : restBew) stand` `  | x == stand  = (erhoeheBewertung (x:y:rest)) : restBew` `  | otherwise  = (x:y:rest) : (erhoeheEinen restBew stand)` Es ist auch sinnvoll, eine Hilfsfunktion `erhoeheBewertung` einzusetzen: `-- erhoeheEinen bewertete stand` `erhoeheEinen [] _  = []` `erhoeheEinen ((x:y:rest) : restBew) stand` `  | x == stand  = (erhoeheBewertung (x:y:rest)) : restBew` `  | otherwise  = (x:y:rest) : (erhoeheEinen restBew stand)` `--erhoeheBewertung bewerteter` `erhoeheBewertung (x:y:_) = x : (y+1) : []` Beim Entfernen von Perlen ist zunächst zu prüfen, ob es überhaupt noch Perlen (*mit der Farbe*) gibt.		3	2
	Insgesamt 50 BE	13	25	12

Hamburg | Behörde für Schule, Familie und Berufsbildung

Name _____ / Kurs-Nr. _____

Schriftliche Abiturprüfung
Schuljahr 2025/2026

Informatik

auf erhöhtem Anforderungsniveau

an allgemeinbildenden gymnasialen Oberstufen

Haupttermin
Mittwoch, 22. April 2026, 09:00 Uhr

Unterlagen für die Prüflinge

Allgemeine Arbeitshinweise

- Überprüfen Sie diese Unterlagen auf Vollständigkeit.
- Schreiben Sie auf alle Prüfungsunterlagen Ihren Namen und zusätzlich auf dieses Deckblatt Ihre Kursnummer.
- Kennzeichnen Sie Ihre Entwurfsblätter (Kladde) und Ihre Reinschrift.

Fachspezifische Arbeitshinweise[1]

- Die Arbeitszeit beträgt **300 Minuten**.
- Eine gesonderte Lese- oder Auswahlzeit wird nicht gewährt.
- Hilfsmittel: Taschenrechner (nicht programmierbar, nicht grafikfähig), Rechtschreibwörterbuch

Aufgabenauswahl

- Sie erhalten **drei** Aufgaben zu unterschiedlichen Schwerpunkten.
- Bearbeiten Sie die **Pflichtaufgabe** und wählen Sie **eine** weitere Aufgabe zur Bearbeitung aus.
- Vermerken Sie auch auf Ihrer Reinschrift, welche Aufgaben Sie ausgewählt und bearbeitet haben.

Bearbeitet wurden die folgenden Aufgaben (Bitte kreuzen Sie an):

I	Objektorientierte Modellierung und Programmierung (Pflicht)	(☐Python ☐ Java)
II	Datensicherheit in verteilten Systemen	(☐ Scheme ☐ Haskell)
III	Intelligente Suchverfahren	(☐ Scheme ☐ Haskell)

[1] Entsprechend der „Richtlinie über die Gewährung von Erleichterungen für neu zugewanderte Schülerinnen, Schüler und Prüflinge bei Sprachschwierigkeiten in der deutschen Sprache“ (MBlSchul Nr. 08, 7. Oktober 2016, S. 60) werden für die betroffenen Prüflinge die folgenden Erleichterungen gewährt:
- Die Bearbeitungszeit wird um 30 Minuten auf **330 Minuten** erhöht.
- Ein nicht-elektronisches Wörterbuch Deutsch – Herkunftssprache / Herkunftssprache – Deutsch wird bereitgestellt.

Bewertung

Jeder Aufgabe sind 50 Bewertungseinheiten (BE) zugeordnet. In allen Teilaufgaben werden nur ganze oder halbe BE vergeben. Insgesamt sind 100 BE erreichbar. Bei der Festlegung von Notenpunkten gilt die folgende Tabelle.

Erbrachte Leistung (in BE)	Notenpunkte
≥ 95	15
≥ 90	14
≥ 85	13
≥ 80	12
≥ 75	11
≥ 70	10
≥ 65	9
≥ 60	8

Erbrachte Leistung (in BE)	Notenpunkte
≥ 55	7
≥ 50	6
≥ 45	5
≥ 40	4
≥ 33	3
≥ 27	2
≥ 20	1
< 20	0

Für die Erteilung der **Note gut** (11 Punkte) ist mindestens erforderlich, dass 75 % der erwarteten Gesamtleistung erbracht werden. Dabei muss die Prüfungsleistung in ihrer Gliederung, in der Gedankenführung, in der Anwendung fachmethodischer Verfahren sowie in der fachsprachlichen Artikulation den Anforderungen voll entsprechen.

Für die Erteilung der **Note ausreichend** (5 Punkte) ist mindestens erforderlich, dass mindestens 45 % der erwarteten Gesamtleistung und über den Anforderungsbereich I hinaus Leistungen in einem weiteren Anforderungsbereich erbracht werden.

Die zwei voneinander unabhängigen Aufgaben der Prüfungsaufgabe werden jeweils mit 50 Bewertungseinheiten bewertet. Die erbrachte Gesamtleistung ergibt sich aus der Summe der Bewertungseinheiten in den beiden Aufgaben.

Schwerwiegende und gehäufte Verstöße gegen die sprachliche Richtigkeit oder gegen die äußere Form führen zu einem Abzug von bis zu zwei Punkten in einfacher Wertung. Mangelhafte Gliederung, Fehler in der Fachsprache, Ungenauigkeiten in Zeichnungen oder unzureichende oder falsche Bezüge zwischen Zeichnungen und Text sind als fachliche Fehler zu werten. Ein Abzug für Verstöße gegen die sprachliche Richtigkeit soll nicht erfolgen, wenn diese bereits Gegenstand der fachspezifischen Bewertungsvorgaben sind.

Aufgabe I: Schiffskarten (Python-Version) 50 BE

Schwerpunkt: Objektorientierte Programmierung und Modellierung

Größere Schiffe senden über das AIS[2] in kurzen Zeitabständen ihre aktuelle Position aus, um Kollisionen mit anderen Schiffen zu vermeiden. Apps wie *Vesselfinder* oder *MarineTraffic* stellen diese Positionen auf Karten dar (vgl. Abb. 1). So kann bspw. ein vom Ufer aus entdecktes Schiff identifiziert werden.

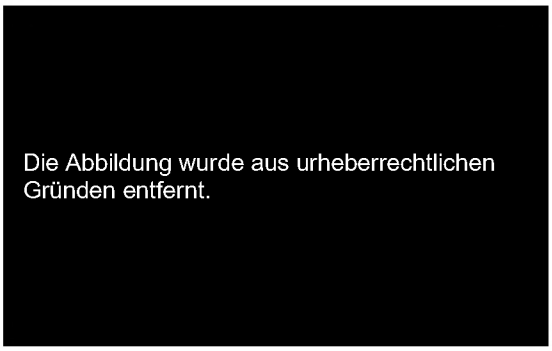


Abbildung 1: Darstellung in einer App. Die Kreise stellen unterschiedliche Schiffe dar. Der Anker steht für den Hafen, der Sendemast für die AIS-Funkstation.

In einer neuen App sollen die Schiffe dargestellt werden, die gerade in Hamburg fahren oder im Hafen liegen. Dabei liegt der Fokus zunächst auf dem Hafengebiet, perspektivisch soll auch der Bereich der Elbe von Geesthacht im Osten Hamburgs bis zur Mündung betrachtet werden. In dem dargestellten Gebiet sollen auch die Häfen und die AIS-Empfangsstationen eingezeichnet werden.

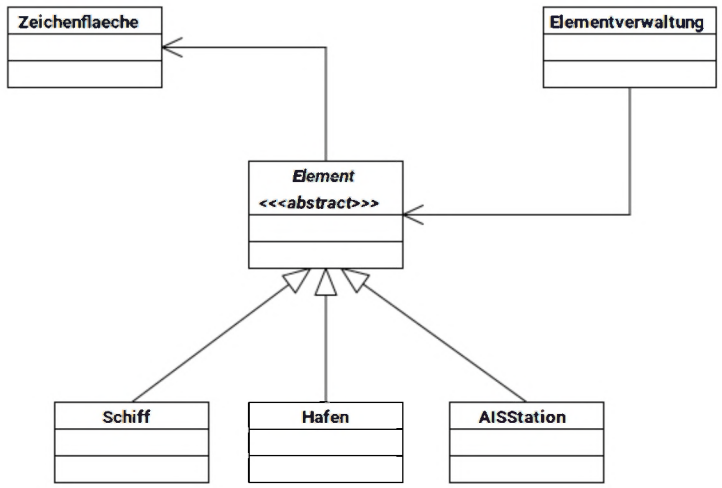


Abbildung 2: Klassendiagramm

I.a • **Geben** Sie **an**, welche Objekte für die Darstellung der `Elemente` auf der Karte in Abb. 1 erzeugt werden müssen.
• **Erläutern** Sie an einem Beispiel aus diesem Projekt, was man unter einer Klasse versteht.

(9 BE)

[2] Das *Automatic Identification System* AIS dient der Sicherheit und Lenkung des Schiffsverkehrs, vgl. Anlage 1.

I.b
- **Erläutern** Sie die Beziehungstypen zwischen den Klassen im Klassendiagramm in Abb. 2.
- **Geben** Sie **an**, woran Sie die unterschiedlichen Beziehungstypen im Programmtext erkennen.
- **Stellen** Sie am Beispiel der Klasse `Element` **dar**, welche Vorteile abstrakte Klassen in der objektorientierten Modellierung haben.

(15 BE)

Unterschiedliche Typen von Schiffen (Handelsschiff, Personenschiff, Tanker, Freizeitboot etc.) sollen auf der Karte in unterschiedlichen Farben dargestellt werden.

Ein fahrendes Schiff soll durch ein Dreieck dargestellt werden. Ein Schiff, das im Hafen oder vor Anker liegt, soll durch einen Kreis symbolisiert werden. Beim fahrenden Schiff soll neben der Position auch die Richtung korrekt in der Karte dargestellt werden. Dafür hat die Klasse Schiff ein Attribut `kurs` vom Typ `int`, in dem der Winkel zwischen Norden und der Fahrtrichtung gespeichert wird. Das darstellende Dreieck wird auf diesen Winkel gedreht und an der richtigen Stelle platziert.

	in Bewegung	gestoppt
Handelsschiff		
Personenschiff		
Tankschiff		
Freizeitboot		

Tabelle 1: Darstellung der verschiedenen Typen von Schiffen in Bewegung und gestoppt.

Zur Umsetzung dieser Anforderung hat die Klasse Schiff ein Attribut `schiffstyp` vom Typ `String` und ein Attribut `inBewegung` vom Typ `boolean`. Alternativ könnten stattdessen Unterklassen für die verschiedenen Schiffstypen von der Klasse `Schiff` abgeleitet werden.

I.c
- **Begründen** Sie, dass beide Varianten eine Lösung für die Anforderung der unterschiedlichen Färbung darstellen.
- **Arbeiten** Sie **heraus**, warum die Unterscheidung fahrender und vor Anker liegender Schiffe nicht über verschiedene Unterklassen geschehen sollte.
- **Erläutern** Sie das Konzept der Polymorphie anhand der Modellierung mit Unterklassen, die unterschiedliche Formen statt unterschiedlicher Farben für die verschiedenen Schiffstypen nutzen.

(7 BE)

Alle Schiffe, die über die Klasse `Zeichenflaeche` auf dem Bildschirm gezeichnet werden sollen, werden dort in der folgenden Datenstruktur gesammelt:

```
self.schiffsliste = []
```

I.d
- **Erläutern** Sie, inwieweit die Wahl dieser Datenstruktur sinnvoll ist. Gehen Sie dabei auf die Datenstruktur Tupel ein.

(3 BE)

Im Hamburger Hafen ist stets nur eine kleine Auswahl aller Schiffe vor Ort, die insgesamt auf der Welt unterwegs sind. Trotzdem kann es vorkommen, dass zeitgleich zwei Schiffe mit demselben Namen hier liegen. So lagen am 20.03.25 zwei Schiffe mit dem Namen *Hamburg Express* nur etwa 2 km voneinander entfernt – der Containerfrachter und ein Segelschiff.

In der Klasse Elementverwaltung werden die aktuell registrierten Schiffe in einer Liste gehalten:

```
self.schiffsliste = []
```

I.e
- **Implementieren** Sie eine Methode, die ein Schiff aus der `schiffsliste` entfernt, wenn erkannt wurde, dass es aus dem von der App abgedeckten geographischen Bereich herausgefahren ist.
- **Erläutern** Sie, wie Sie in der Implementierung erreichen, dass das richtige Schiff aus der `schiffsliste` entfernt wird.

(6 BE)

Die Position eines Schiffs wird mit Hilfe von Satelliten, z. B. des GPS-Systems[3], ermittelt. Die dafür genutzte Antenne ist an einem bestimmten Punkt an Bord eingebaut. Gerade Containerschiffe sind oft so groß, dass diese Position einen Einfluss auf die Positionierung auf der Karte hat und entscheidend dafür sein kann, ob zwei Schiffe aneinander vorbeifahren oder kollidieren. Die *Hamburg Express* ist mit 399 m Länge bspw. so lang, dass der Michel, die größte Kirche in Hamburg, dreimal auf das Schiff passen würde.

In der App soll daher neben den Kreisen und den Dreiecken bei entsprechender Zoomstufe auch der grobe Umriss großer Schiffe dargestellt werden.

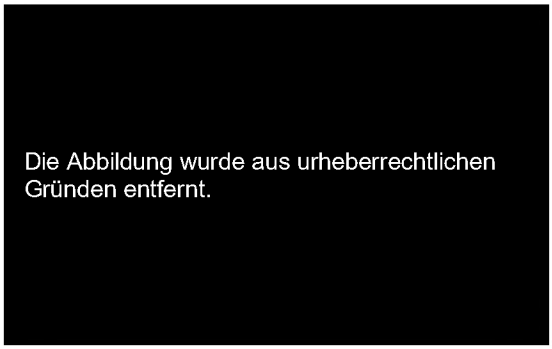


Abbildung 3: Kartenausschnitt mit Schiffen und eingezeichneter Ausdehnung um die gemeldete Position herum

I.f
- **Analysieren** Sie, um welche Attribute Sie die Klasse `Schiff` erweitern würden, um die Größe des Schiffs und die Lage des Schiffs um die ausgesendete Position herum darzustellen.
- **Erklären** Sie, wie dabei eine korrekte Positionierung erreicht werden kann, wenn der Kreis immer in derselben Größe angezeigt wird, die Größe der Darstellung des Schiffs sich aber abhängig von der Zoomstufe ändert.

(6 BE)

[3] *Global Positioning System*, s. Anlage 2

Für die Planung der Abläufe im Hafen ist eine Übersicht über die Frachtschiffe erforderlich, die im Hafen liegen und auf der Elbe bzw. der Nordsee unterwegs sind. Dazu sollen Listen erzeugt werden können, die die empfangenen Meldungen mehrerer AIS-Stationen sammeln. Westlich von Hamburg sollen vier Stationen bis zur Nordsee (Wedel, Rhinplate, Brunsbüttel und Cuxhaven), östlich von Hamburg sechs Stationen bis Magdeburg und in Hamburg die drei Stationen im Stadtgebiet zu AIS-Gebieten zusammengefasst werden.

I.g
- **Entwerfen** Sie eine Erweiterung des Klassendiagramms um AIS-Gebiete, die jeweils bestimmte AIS-Stationen enthalten.
- **Erklären** Sie, wie die aktuelle Liste von Schiffen in einem bestimmten Gebiet erzeugt wird, wenn die entsprechende Methode eines AIS-Gebiets aufgerufen wird.

(4 BE)

Anlage zur Aufgabe Schiffskarten (Python-Version)

Anlage 1 AIS

Das *Automatic Identification System* AIS dient der Sicherheit und Lenkung des Schiffsverkehrs. Alle Schiffe senden ihre Position, ihren Kurs[4] und ihre Geschwindigkeit sowie ihren Namen, ihre Größe und die MMSI-Nummer[5] – eine weltweit eindeutige 9-stellige Nummer. Bei größeren Schiffen werden auch Angaben über die Ladung und den Zielort ausgesendet. An Bord werden die Aussendungen von anderen Schiffen in der Nähe empfangen. Diese Information wird zur Kollisionsvermeidung genutzt. AIS-Basisstationen an Land erfassen mit Hilfe der Meldungen der Schiffe den Schiffsverkehr im von ihnen abgedeckten Gebiet. Die Reichweite des Signals zwischen zwei Schiffen beträgt ca. 37 km. Entlang der Elbe zwischen Hamburg und der Mündung gibt es neun solcher Stationen, davon drei im Hamburger Stadtgebiet, davon eine in Neumühlen.

In der Karte wird hierfür die folgende Darstellung verwendet: 

Anlage 2 GPS

Das *Global Positioning System* erlaubt es, auf der ganzen Erde jederzeit wetterunabhängig die eigene Position zu bestimmen. Dafür werden Satellitensignale ausgewertet.

[4] Fachbegriff für die „Fahrtrichtung“ eines Schiffs
[5] *Maritime Mobile Service Identity*: Der Containerfrachter Hamburg Express hat bspw. die MMSI-Nummer 211123000.

Aufgabe I: Schiffskarten (Java-Version) 50 BE

Schwerpunkt: Objektorientierte Programmierung und Modellierung

Größere Schiffe senden über das AIS-System[6] in kurzen Zeitabständen ihre aktuelle Position aus, um Kollisionen mit anderen Schiffen zu vermeiden. Apps wie *Vesselfinder* oder *MarineTraffic* stellen diese Positionen auf Karten dar (vgl. Abb. 1). So kann bspw. ein vom Ufer aus entdecktes Schiff identifiziert werden.

Abbildung 1: Darstellung in einer App. Die Kreise stellen unterschiedliche Schiffe dar. Der Anker steht für den Hafen, der Sendemast für die AIS-Funkstation.

In einer neuen App sollen die Schiffe dargestellt werden, die gerade in Hamburg fahren oder im Hafen liegen. Dabei liegt der Fokus zunächst auf dem Hafengebiet, perspektivisch soll auch der Bereich der Elbe von Geesthacht im Osten Hamburgs bis zur Mündung betrachtet werden. In dem dargestellten Gebiet sollen auch die Häfen und die AIS-Empfangsstationen eingezeichnet werden.

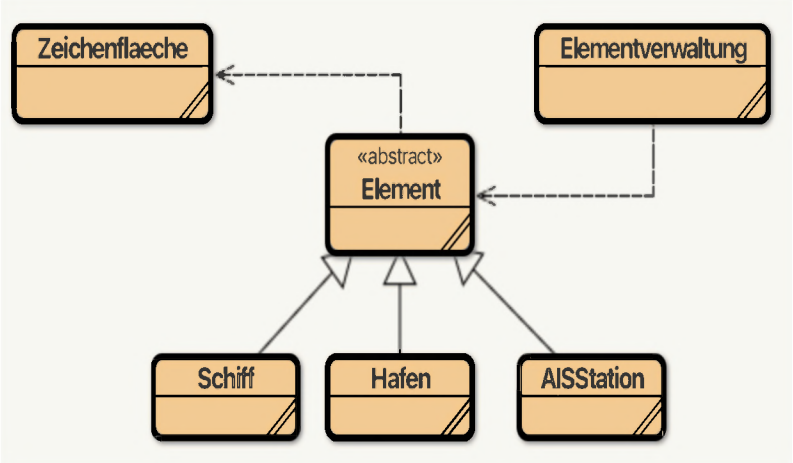


Abbildung 2: Klassendiagramm

[6] Das *Automatic Identification System* AIS dient der Sicherheit und Lenkung des Schiffsverkehrs, vgl. Anlage 1.

I.a
- **Geben** Sie **an**, welche Objekte für die Darstellung der `Elemente` auf der Karte in Abb. 1 erzeugt werden müssen.
- **Erläutern** Sie an einem Beispiel aus diesem Projekt, was man unter einer Klasse versteht.

(9 BE)

I.b
- **Erläutern** Sie die Beziehungstypen zwischen den Klassen im Klassendiagramm in Abb. 2.
- **Geben** Sie **an**, woran Sie die unterschiedlichen Beziehungstypen im Programmtext erkennen.
- **Stellen** Sie am Beispiel der Klasse `Element` **dar**, welche Vorteile abstrakte Klassen in der objektorientierten Modellierung haben.

(15 BE)

Unterschiedliche Typen von Schiffen (Handelsschiff, Personenschiff, Tanker, Freizeitboot etc.) sollen auf der Karte in unterschiedlichen Farben dargestellt werden.

Ein fahrendes Schiff soll durch ein Dreieck dargestellt werden. Ein Schiff, das im Hafen oder vor Anker liegt, soll durch einen Kreis symbolisiert werden. Beim fahrenden Schiff soll neben der Position auch die Richtung korrekt in der Karte dargestellt werden. Dafür hat die Klasse Schiff ein Attribut `kurs` vom Typ `int`, in dem der Winkel zwischen Norden und der Fahrtrichtung gespeichert wird. Das darstellende Dreieck wird auf diesen Winkel gedreht und an der richtigen Stelle platziert.

	in Bewegung	gestoppt
Handelsschiff		
Personenschiff		
Tankschiff		
Freizeitboot		

Tabelle 1: Darstellung der verschiedenen Typen von Schiffen in Bewegung und gestoppt.

Zur Umsetzung dieser Anforderung hat die Klasse Schiff ein Attribut `schiffstyp` vom Typ `String` und ein Attribut `inBewegung` vom Typ `boolean`. Alternativ könnten stattdessen Unterklassen für die verschiedenen Schiffstypen von der Klasse `Schiff` abgeleitet werden.

I.c
- **Begründen** Sie, dass beide Varianten eine Lösung für die Anforderung der unterschiedlichen Färbung darstellen.
- **Arbeiten** Sie **heraus**, warum die Unterscheidung fahrender und vor Anker liegender Schiffe nicht über verschiedene Unterklassen geschehen sollte.
- **Erläutern** Sie das Konzept der Polymorphie anhand der Modellierung mit Unterklassen, die unterschiedliche Formen statt unterschiedlicher Farben für die verschiedenen Schiffstypen nutzen.

(7 BE)

Alle Schiffe, die über die Klasse `Zeichenflaeche` auf dem Bildschirm gezeichnet werden sollen, werden dort in der folgenden Datenstruktur gesammelt:

```
private Schiff[] schiffe;
```

I.d • **Erläutern** Sie, inwieweit die Wahl dieser Datenstruktur sinnvoll ist. Gehen Sie dabei auf die Datenstruktur ArrayList ein.

(3 BE)

Im Hamburger Hafen ist stets nur eine kleine Auswahl aller Schiffe vor Ort, die insgesamt auf der Welt unterwegs sind. Trotzdem kann es vorkommen, dass zeitgleich zwei Schiffe mit demselben Namen hier liegen. So lagen am 20.03.25 zwei Schiffe mit dem Namen *Hamburg Express* nur etwa 2 km voneinander entfernt – der Containerfrachter und ein Segelschiff.

In der Klasse Elementverwaltung werden die aktuell registrierten Schiffe, anders als in Aufgabe d, in einer ArrayList gehalten:

```
private ArrayList<Schiffe> schiffsliste;
```

I.e • **Implementieren** Sie eine Methode, die ein Schiff aus der `schiffsliste` entfernt, wenn erkannt wurde, dass es aus dem von der App abgedeckten geographischen Bereich herausgefahren ist.
• **Erläutern** Sie, wie Sie in der Implementierung erreichen, dass das richtige Schiff aus der `schiffsliste` entfernt wird.

(6 BE)

Die Position eines Schiffs wird mit Hilfe von Satelliten, z. B. des GPS-Systems[7], ermittelt. Die dafür genutzte Antenne ist an einem bestimmten Punkt an Bord eingebaut. Gerade Containerschiffe sind oft so groß, dass diese Position einen Einfluss auf die Positionierung auf der Karte hat und entscheidend dafür sein kann, ob zwei Schiffe aneinander vorbeifahren oder kollidieren. Die *Hamburg Express* ist mit 399 m Länge bspw. so lang, dass der Michel, die größte Kirche in Hamburg, dreimal auf das Schiff passen würde.

In der App soll daher neben den Kreisen und den Dreiecken bei entsprechender Zoomstufe auch der grobe Umriss großer Schiffe dargestellt werden.

Abbildung 3: Kartenausschnitt mit Schiffen und eingezeichneter Ausdehnung um die gemeldete Position herum

[7] *Global Positioning System*, s. Anlage 2

I.f • **Analysieren** Sie, um welche Attribute Sie die Klasse `Schiff` erweitern würden, um die Größe des Schiffs und die Lage des Schiffs um die ausgesendete Position herum darzustellen.
• **Erklären** Sie, wie dabei eine korrekte Positionierung erreicht werden kann, wenn der Kreis immer in derselben Größe angezeigt wird, die Größe der Darstellung des Schiffs sich aber abhängig von der Zoomstufe ändert.

(6 BE)

Für die Planung der Abläufe im Hafen ist eine Übersicht über die Frachtschiffe erforderlich, die im Hafen liegen und auf der Elbe bzw. der Nordsee unterwegs sind. Dazu sollen Listen erzeugt werden können, die die empfangenen Meldungen mehrerer AIS-Stationen sammeln. Westlich von Hamburg sollen vier Stationen bis zur Nordsee (Wedel, Rhinplate, Brunsbüttel und Cuxhaven), östlich von Hamburg sechs Stationen bis Magdeburg und in Hamburg die drei Stationen im Stadtgebiet zu AIS-Gebieten zusammengefasst werden.

I.g • **Entwerfen** Sie eine Erweiterung des Klassendiagramms um AIS-Gebiete, die jeweils bestimmte AIS-Stationen enthalten.
• **Erklären** Sie, wie die aktuelle Liste von Schiffen in einem bestimmten Gebiet erzeugt wird, wenn die entsprechende Methode eines AIS-Gebiets aufgerufen wird.

(4 BE)

Anlage zur Aufgabe Schiffskarten (Java-Version)

Anlage 1 AIS

Das *Automatic Identification System* AIS dient der Sicherheit und Lenkung des Schiffsverkehrs. Alle Schiffe senden ihre Position, ihren Kurs[8] und ihre Geschwindigkeit sowie ihren Namen, ihre Größe und die MMSI-Nummer[9] – eine weltweit eindeutige 9-stellige Nummer. Bei größeren Schiffen werden auch Angaben über die Ladung und den Zielort ausgesendet. An Bord werden die Aussendungen von anderen Schiffen in der Nähe empfangen. Diese Information wird zur Kollisionsvermeidung genutzt. AIS-Basisstationen an Land erfassen mit Hilfe der Meldungen der Schiffe den Schiffsverkehr im von ihnen abgedeckten Gebiet. Die Reichweite des Signals zwischen zwei Schiffen beträgt ca. 37 km. Entlang der Elbe zwischen Hamburg und der Mündung gibt es neun solcher Stationen, davon drei im Hamburger Stadtgebiet, davon eine in Neumühlen.

In der Karte wird hierfür die folgende Darstellung verwendet: 

Anlage 2 GPS

Das *Global Positioning System* erlaubt es, auf der ganzen Erde jederzeit wetterunabhängig die eigene Position zu bestimmen. Dafür werden Satellitensignale ausgewertet.

[8] Fachbegriff für die „Fahrtrichtung“ eines Schiffs
[9] *Maritime Mobile Service Identity*: Der Containerfrachter Hamburg Express hat bspw. die MMSI-Nummer 211123000.

Aufgabe II: Bitcoin (Scheme-Version) 50 BE

Schwerpunkt: Datensicherheit in verteilten Systemen

Die Bitcoin-Blockchain ist eine einzigartige, fortlaufende Kette von Datenblöcken. Jeder Block enthält etwa 1.000 bis 3.000 Transaktionsdaten (Käufe und Verkäufe von Bitcoin), vergleichbar mit Banküberweisungen. Das Besondere an dieser Struktur ist die Art der Verknüpfung: Die Blöcke sind durch spezielle Hashwerte (kurz Hashes[10]) miteinander verbunden, wobei der Hash jedes Blocks in die Berechnung des Hashes des nachfolgenden Blocks einfließt. Diese Verkettung macht nachträgliche Manipulationen der Transaktionsdaten praktisch unmöglich.

Im Gegensatz zur sonst üblichen einmaligen Berechnung eines Hashs für einen Datenblock muss bei der Bitcoin-Blockchain ein spezieller Hash gefunden werden, der in der hexadezimalen Darstellung eine bestimmte Anzahl führender Nullen aufweist. Diese Nullen kommen in der hexadezimalen Darstellung (als String) dadurch zustande, dass eine feste Länge an Zeichen vorgegeben wird und, wenn die umgerechnete Dezimalzahl weniger Hexadezimal-Stellen aufweist, der hexadezimale String einfach vorne mit Nullen aufgefüllt wird. Diese Anzahl führender Nullen für einen „gültigen“ Hash wird durch den Parameter *difficulty* (Schwierigkeit) vorgegeben. Da es nicht möglich ist, vorherzuberechnen, wie man die Daten im Block ändern muss, um einen bestimmten Hash zu erhalten, kann dieses Problem nur durch Ausprobieren gelöst werden. Je mehr führende Nullen gefordert sind, desto höher ist die Schwierigkeit und desto größer der erforderliche Rechenaufwand.

Der Prozess des „Schürfens" (m*ining*) besteht darin, einem Block eine Zahl („Nonce" – „**N**umber only used **once**“) hinzuzufügen und diese solange zu verändern, bis ein Hash mit der geforderten Anzahl führender Nullen gefunden wird. Diese rechenintensive Aufgabe ist der Kern des Bitcoin-Mining-Prozesses.

Jeder neue Block dieser Bitcoin-Blockchain enthält unter anderem:

- **Transaktionsdaten**: Daten, wie z. B. „Alice sendet 5 BTC an Bertram“.
- **„Previous“ Block-Hash**: Der Hash des vorherigen Blocks, der die Kette sichert.
- **Nonce**: Eine einfache Zahl, die den Transaktionsdaten am Ende hinzugefügt wird. Sie wird während des Mining-Prozesses hochgezählt, um einen passenden, neuen Hash zu finden.
- **„New“ Block-Hash**: Der eigene Hash des Blocks, der durch das Mining neu berechnet wurde und für den die Miner die „Belohnung” in Bitcoin bekommen.

II.a Hashing ist ein wesentlicher Bestandteil der Blockchain.
• **Geben** Sie in diesem Kontext **an**, ob durch das Hashing die Authentizität, die Integrität und/oder die Vertraulichkeit der in der Blockchain verwalteten Daten sichergestellt werden.
• **Erläutern** Sie, warum die Daten der Blockchain in Bezug auf diese/-n Punkt/-e „sicher“ sind.

(11 BE)

In Anlage 1 finden Sie die Funktion `einfacherHash`, die ein vereinfachtes Hashing-Verfahren implementiert. Diese Funktion berechnet einen Hashwert in dezimaler Darstellung. Zum Beispiel ergibt der Aufruf `(einfacherHash "Hallo, dies ist ein Test" 31)` den Wert `6266346`.

II.b • **Beschreiben** Sie, wie dieses einfache Hash-Verfahren zu einem Hash mit begrenzter Länge führt.
• **Analysieren** Sie die Funktionsweise von `einfacherHash`.

(13 BE)

[10] **Hashwert**: Ein Hashwert (auch Hash genannt) ist eine eindeutige Zeichenfolge fester Länge, die aus beliebigen Daten berechnet wird – wie ein digitaler Fingerabdruck. Die Berechnung des Hashes ist schnell und einfach und die Rückrechnung vom Hash zu den ursprünglichen Daten ist praktisch unmöglich (Einwegfunktion).

Der durch die Blockchain-Vorgaben erzwungene Rechenaufwand, um einen „passenden Hash“ zu finden („schürfen“), schafft einen Wettbewerb zwischen den „Minern", die darum wetteifern, den jeweils nächsten Block mit einem passenden Hash zu finden. Erst wenn dieser gefunden ist, kann der Block zur Bitcoin-Kette hinzugefügt werden, da der Hash des Vorgängerblocks Teil des neuen Hashs ist. Diese Aufgabe erfordert je nach festgelegter Schwierigkeit (bestimmt durch die Anzahl der führenden Nullen) erhebliche Rechenleistung, da eine Lösung nur durch das Ausprobieren verschiedener Nonces (zusätzliche Zahlenwerte) gefunden werden kann.
Die Miner investieren diese Rechenleistung aufgrund der in Aussicht gestellten Belohnung in Form von Bitcoin. Mit höherer Rechenkapazität kann ein Miner seine Chancen verbessern, einen gültigen Hash zuerst zu finden und somit in den Genuss der Belohnung zu kommen.

Beispiel (vereinfacht): Es soll ein Hash mit sechs hexadezimalen Stellen erzeugt werden, wobei als Schwierigkeitsgrad mindestens zwei führende Nullen erforderlich sind. Der Hash wird mithilfe der Funktion `mineBlock` in **Anlage 1** berechnet:

Daten: "15.05.2025: Alice an Paul 0.003 BTC“
Previous Hash (Hex): "00c2b0"
Nonce: 0

Beim Ausprobieren verschiedener Nonces kommt man z. B. auf folgende Werte:

Nonce Wert 218	→	Hash (Dezimal):	1323898	→	Hexadezimal: 014337
Nonce Wert 219	→	Hash (Dezimal):	40196	→	Hexadezimal: **009d04**

Die Berechnung ist abgeschlossen, da mit dem Hash **009d04** die vorgegebene Schwierigkeit (mindestens zwei führende Nullen in der hexadezimalen Darstellung) erfüllt ist.

II.c **Erklären** Sie anhand dieses Beispiels, warum es beim Hashing so wichtig ist, dass ähnliche Daten keine ähnlichen Hashes erzeugen.

(5 BE)

II.d • **Erläutern** Sie anhand der Funktion `mineBlock` (siehe Anlage I), wie ein neuer Block mit einem gültigen neuen Hash versehen wird, welcher der vorgegebenen Schwierigkeit entspricht.
Für diese Aufgabe ist es nicht nötig, im Detail auf die Funktionsweise der Funktion `einfacherHash` einzugehen. Es reicht zu verstehen, dass diese Funktion aus verschiedenen, übergebenen Parametern einen Hash fester Länge berechnet.
• **Implementieren** Sie die noch fehlende Funktion:

```
(stringReferenz hexString position)
```

Die Beschreibung der Aufgabe der Funktion `stringReferenz` finden Sie im Anlage I als Kommentar oberhalb der Funktion.

(17 BE)

Der gefundene, passende Hash, der am Ende den neuen Block verifiziert, ist das Ergebnis des „Proof-of-Work-Prozesses“ (Nachweis-von-Arbeit-Prozesses) .

II.e **Begründen** Sie, dass „Proof-of-Work“ ein passender Name für diesen Prozess ist.

(4 BE)

Anlage zur Aufgabe Bitcoin (Scheme-Version)

Anlage 1 Hashingprojekt

Um einen neuen Block der Blockchain mit dem passenden hexadezimalen Hash (mit entsprechend vielen Nullen vorne) hinzufügen zu können, ist folgender Code vorgegeben:

Anmerkungen

- (number->string `zahl` 16) rechnet eine dezimale Darstellung einer Zahl `zahl` in eine hexadezimale Darstellung (16er-System) um.
- (string-length „xxx“) berechnet die Länge des Strings „xxx“
- (string-append „a” „xxx”) fügt die beiden angegebenen Strings zusammen: „axxx”.

```
(define (einfacherHash daten faktor)
  (let ([MOD 16777216])
    (modulo (hashHilf (string->list daten) 0 1 0 faktor MOD) MOD)))

(define (hashHilf chars index multi summe faktor MOD)
  (cond
    [(null? chars) summe]
    [else
     (hashHilf
      (rest chars)
      (+ index 1)
      (modulo (* multi faktor) MOD)
      (+ summe (* (char->integer (first chars))
                  (+ multi index)))
      faktor
      MOD)]))
; Test
;(einfacherHash "Hallo, dies ist ein Test" 31); -> 6266346


#| HINWEIS:
Die Funktion stringReferenz soll - falls der übergebene String (hexString)
lang genug ist - das Zeichen an der angegebenen position zurückgeben,
andernfalls das Zeichen #\? .
Diese Funktion muss von Ihnen noch implementiert werden! Siehe Aufgabe
II.d. |#

(define (stringReferenz hexString position)
  ...)

(define (int->hexString dezimalzahl laenge)
  (fuelleString (number->string dezimalzahl 16) laenge))

(define (fuelleString str noetigeLaenge)
  (cond ((>= (string-length str) noetigeLaenge) str)
        (else (fuelleString (string-append "0" str) noetigeLaenge))))

(define (hashTrifftSchwierigkeit? hashWert schwierigkeit maxStellenHash)
  (checkPrefix?
   (int->hexString hashWert maxStellenHash)
   0
   schwierigkeit))
```

```
(define (checkPrefix? hexString position schwierigkeit)
  (cond ((>= position schwierigkeit) #t)
        (else (cond ((equal? (stringReferenz hexString position) #\0)
                     (checkPrefix? hexString
                                   (+ position 1)
                                   schwierigkeit))
                    (else  #f)))))

 (define (mineBlock daten prevHash schwierigkeit maxStellenHash faktor)
  (mineBlockHilf daten
                 prevHash
                 schwierigkeit
                 0
                 (einfacherHash (string-append daten prevHash "0")
                                faktor)
                 maxStellenHash
                 faktor))


(define (mineBlockHilf daten prevHash schwierigkeit nonce hashWert
                       maxStellenHash faktor)
  (cond ((hashTrifftSchwierigkeit? hashWert schwierigkeit maxStellenHash)
         (list nonce
               hashWert
               (int->hexString hashWert maxStellenHash)
               daten
               prevHash))
        (else
         (mineBlockHilf
          daten
          prevHash
          schwierigkeit
          (+ nonce 1)
          (einfacherHash
           (string-append
            daten
            prevHash
            (number->string (+ nonce 1)))
           faktor)
          maxStellenHash
          faktor  ))))

(mineBlock "Hallo, dies ist ein Test" "006dAA" 2 6 31)
; -> (471 17084 "0042bc" "Hallo, dies ist ein Test" "006dAA")
```

Aufgabe II: Bitcoin (Haskell-Version) 50 BE

Schwerpunkt: Datensicherheit in verteilten Systemen

Die Bitcoin-Blockchain ist eine einzigartige, fortlaufende Kette von Datenblöcken. Jeder Block enthält etwa 1.000 bis 3.000 Transaktionsdaten (Käufe und Verkäufe von Bitcoin), vergleichbar mit Banküberweisungen. Das Besondere an dieser Struktur ist die Art der Verknüpfung: Die Blöcke sind durch spezielle Hashwerte (kurz Hashes[11]) miteinander verbunden, wobei der Hash jedes Blocks in die Berechnung des Hashes des nachfolgenden Blocks einfließt. Diese Verkettung macht nachträgliche Manipulationen der Transaktionsdaten praktisch unmöglich.

Im Gegensatz zur sonst üblichen einmaligen Berechnung eines Hashs für einen Datenblock muss bei der Bitcoin-Blockchain ein spezieller Hash gefunden werden, der in der hexadezimalen Darstellung eine bestimmte Anzahl führender Nullen aufweist. Diese Nullen kommen in der hexadezimalen Darstellung (als String) dadurch zustande, dass eine feste Länge an Zeichen vorgegeben wird und, wenn die umgerechnete Dezimalzahl weniger Hexadezimal-Stellen aufweist, der hexadezimale String einfach vorne mit Nullen aufgefüllt wird. Diese Anzahl führender Nullen für einen „gültigen“ Hash wird durch den Parameter *difficulty* (Schwierigkeit) vorgegeben. Da es nicht möglich ist, vorherzuberechnen, wie man die Daten im Block ändern muss, um einen bestimmten Hash zu erhalten, kann dieses Problem nur durch Ausprobieren gelöst werden. Je mehr führende Nullen gefordert sind, desto höher ist die Schwierigkeit und desto größer der erforderliche Rechenaufwand.

Der Prozess des „Schürfens“ (*mining*) besteht darin, einem Block eine Zahl („Nonce“ – „**N**umber only used **once**“) hinzuzufügen und diese solange zu verändern, bis ein Hash mit der geforderten Anzahl führender Nullen gefunden wird. Diese rechenintensive Aufgabe ist der Kern des Bitcoin-Mining-Prozesses.

Jeder neue Block dieser Bitcoin-Blockchain enthält unter anderem:

- **Transaktionsdaten**: Daten, wie z. B. „Alice sendet 5 BTC an Bertram“.
- **„Previous“ Block-Hash**: Der Hash des vorherigen Blocks, der die Kette sichert.
- **Nonce**: Eine einfache Zahl, die den Transaktionsdaten am Ende hinzugefügt wird. Sie wird während des Mining-Prozesses hochgezählt, um einen passenden, neuen Hash zu finden.
- **„New“ Block-Hash**: Der eigene Hash des Blocks, der durch das Mining neu berechnet wurde und für den die Miner die „Belohnung“ in Bitcoin bekommen.

II.a Hashing ist ein wesentlicher Bestandteil der Blockchain.
• **Geben** Sie in diesem Kontext **an**, ob durch das Hashing die Authentizität, die Integrität und/oder die Vertraulichkeit der in der Blockchain verwalteten Daten sichergestellt werden.
• **Erläutern** Sie warum die Daten der Blockchain in Bezug auf diese/n Punkt/e „sicher“ sind.

(11 BE)

In Anlage 1 finden Sie die Funktion `einfacherHash`, die ein vereinfachtes Hashing-Verfahren implementiert. Diese Funktion berechnet einen Hashwert in dezimaler Darstellung. Zum Beispiel ergibt der Aufruf

`einfacherHash "Hallo, dies ist ein Test" 31` den Wert `6266346`.

II.b • **Beschreiben** Sie, wie dieses einfache Hash-Verfahren zu einem Hash mit begrenzter Länge führt.
• **Analysieren** Sie die Funktionsweise von `einfacherHash`.

(13 BE)

Der durch die Blockchain-Vorgaben erzwungene Rechenaufwand, um einen „passenden Hash“ zu finden („schürfen“), schafft einen Wettbewerb zwischen den „Minern“, die darum wetteifern, den

[11] **Hashwert**: Ein Hashwert (auch Hash genannt) ist eine eindeutige Zeichenfolge fester Länge, die aus beliebigen Daten berechnet wird - wie ein digitaler Fingerabdruck. Die Berechnung des Hashs ist schnell und einfach und die Rückrechnung vom Hash zu den ursprünglichen Daten ist praktisch unmöglich (Einwegfunktion).

jeweils nächsten Block mit einem passenden Hash zu finden. Erst wenn dieser gefunden ist, kann der Block zur Bitcoin-Kette hinzugefügt werden, da der Hash des Vorgängerblocks Teil des neuen Hashs ist. Diese Aufgabe erfordert je nach festgelegter Schwierigkeit (bestimmt durch die Anzahl der führenden Nullen) erhebliche Rechenleistung, da eine Lösung nur durch das Ausprobieren verschiedener Nonces (zusätzliche Zahlenwerte) gefunden werden kann.
Die Miner investieren diese Rechenleistung aufgrund der in Aussicht gestellten Belohnung in Form von Bitcoin. Mit höherer Rechenkapazität kann ein Miner seine Chancen verbessern, einen gültigen Hash zuerst zu finden und somit in den Genuss der Belohnung zu kommen.

Beispiel (vereinfacht): Es soll ein Hash mit sechs hexadezimalen Stellen erzeugt werden, wobei als Schwierigkeitsgrad mindestens zwei führende Nullen erforderlich sind. Der Hash wird mithilfe der Funktion `mineBlock` in **Anlage 1** berechnet:

Daten: "15.05.2025: Alice an Paul 0.003 BTC“
Previous Hash (Hex): "00c2b0"
Nonce: 0

Beim Ausprobieren verschiedener Nonces kommt man z. B. auf folgende Werte:

Nonce Wert 218	→ Hash (Dezimal):	1323898	→	Hexadezimal: 014337
Nonce Wert 219	→ Hash (Dezimal):	40196	→	Hexadezimal: **009d04**

Die Berechnung ist abgeschlossen, da mit dem Hash **009d04** die vorgegebene Schwierigkeit (mindestens zwei führende Nullen in der hexadezimalen Darstellung) erfüllt ist.

II.c **Erklären** Sie anhand dieses Beispiels, warum es beim Hashing so wichtig ist, dass ähnliche Daten keine ähnlichen Hashes erzeugen.

(5 BE)

II.d • **Erläutern** Sie anhand der Funktion `mineBlock` (siehe Anlage 1), wie ein neuer Block mit einem gültigen neuen Hash versehen wird, welcher der vorgegebenen Schwierigkeit entspricht.
Für diese Aufgabe ist es nicht nötig, im Detail auf die Funktionsweise der Funktion `einfacherHash` einzugehen. Es reicht zu verstehen, dass `einfacherHash` aus verschiedenen, übergebenen Parametern einen Hash fester Länge berechnet.

• **Implementieren** Sie die noch fehlende Funktion:

```
stringReferenz hexString position
```

Die Beschreibung der Aufgabe der Funktion `stringReferenz` finden Sie im Anlage 1 als Kommentar oberhalb der Funktion.

(17 BE)

Der gefundene, passende Hash, der am Ende den neuen Block verifiziert, ist das Ergebnis des „Proof-of-Work-Prozesses” (Nachweis-von-Arbeit-Prozesses).

II.e **Begründen** Sie, dass „Proof-of-Work“ ein passender Name für diesen Prozess ist.

(4 BE)

Anlage zur Aufgabe Bitcoin (**Haskell**-Version)

Anlage 1 Hashingprojekt

Um einen neuen Block der Blockchain mit dem passenden hexadezimalen Hash (mit entsprechend vielen Nullen vorne) hinzufügen zu können, ist folgender Code vorgegeben:

Anmerkungen

- `showHex` rechnet eine dezimale Darstellung einer Zahl in eine hexadezimale Darstellung (16er-System) um.
- length berechnet die Länge eines Strings.
- Der Operator `++` fügt die angegebenen Strings zusammen: `"a"++"xxx"` liefert `"axxx"`.
- Der Aufruf:

```
mineBlock "Hallo, dies ist ein Test" "006dAA" 2 6 31
```

liefert

```
(471, 17084, "0042bc", "Hallo, dies ist ein Test", "006dAA")
```

```
import Numeric (showHex)
import Data.Char (ord)

festerModul = 16777216

einfacherHash daten faktor = (hashHilf daten 0 1 0 faktor festerModul) `mod` festerModul

--hashHilf chars index multi summe faktor modul
hashHilf []     _     _     summe _      = summe
hashHilf (c:cs) index multi summe faktor modul
  = hashHilf cs (index+1) ((multi*faktor) `mod` modul) (summe + ((ord c)*(multi+index))) faktor modul

{- Hinweis: Die Funktion stringReferenz soll - falls der übergebene String (hexString) lang genug ist - das
Zeichen an der angegebenen positionzurückgeben, andernfalls das Zeichen '?'.
Diese Funktion muss von Ihnen noch implementiert werden, siehe Aufgabe II.d. -}
stringReferenz :: String -> Int -> Char
stringReferenz hexString position
  …

intToHexString dezimalzahl laenge = fuelleString (showHex dezimalzahl "") laenge

fuelleString str noetigeLaenge
  | (length str) >= noetigeLaenge    = str
  | otherwise                        = fuelleString ("0"++str) noetigeLaenge

hashTrifftSchwierigkeit hashWert schwierigkeit maxStellenHash
  = checkPrefix (intToHexString hashWert maxStellenHash) 0 schwierigkeit

checkPrefix hexString position schwierigkeit
  | position >= schwierigkeit                  = True
  | (stringReferenz hexString position) == '0' = checkPrefix hexString (position+1) schwierigkeit
  | otherwise                                  = False

mineBlock daten prevHash schwierigkeit maxStellenHash faktor
  = mineBlockHilf daten prevHash schwierigkeit 0 (einfacherHash (daten++prevHash++"0") faktor) maxStellenHash faktor

mineBlockHilf daten prevHash schwierigkeit nonce hashWert maxStellenHash faktor
  | hashTrifftSchwierigkeit hashWert schwierigkeit maxStellenHash
    = (nonce, hashWert, (intToHexString hashWert maxStellenHash), daten, prevHash)
  | otherwise
    = mineBlockHilf daten prevHash schwierigkeit (nonce+1) (einfacherHash (daten++prevHash++
      (show (nonce+1))) faktor) maxStellenHash faktor
```

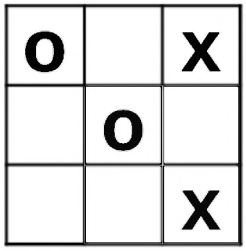
Aufgabe III: MENACE (Scheme-Version) 50 BE

Schwerpunkt: Intelligente Suchverfahren

Unter dem Namen MENACE hat Donald Michie in den 1960er Jahren eine Idee entwickelt, bei der er, noch ohne einen Computer einzusetzen, das Vorgehen moderner KI-Anwendungen vorwegnahm.

(*Text dazu und zu Tic-Tac-Toe siehe Anlagen 1 und 2*)

Für die in dem Text beschriebene Lernphase sollen in dieser Aufgabe einige Teilschritte modelliert werden.



Teilaufgaben

III.a
- **Beschreiben** Sie die folgende angegebene Datenstruktur zum Verwalten des jeweiligen Spielstands von Tic-Tac-Toe. Verwenden Sie dazu das oben dargestellte Bild als Beispiel. Dort hat Spieler 1 (Kreise) bereits zweimal gesetzt und Spieler 2 (Kreuze) ebenfalls, so dass Spieler 1 als nächster setzen kann.

```
'(("o"  " "  "x") (" "  "o"  " ") (" "  " "  "x"))
```

- **Vergleichen** Sie diese mit einer Liste der folgenden Form:

```
'("o"  " "  "x"  " "  "o"  " "  " "  " "  "x")
```

(8 BE)

III.b
- **Beschreiben** Sie, wie man mit Hilfe von Breitensuche alle möglichen Spielzustände nach dem Setzen der ersten drei Zeichen (also zum Beispiel Kreis, Kreuz, Kreis) bestimmen kann.
- **Bestimmen** Sie dabei auch die Anzahl der möglichen verschiedenen Abfolgen beim Setzen der drei Zeichen.
- **Begründen** Sie, dass sich viele dieser Spielzustände prinzipiell (also im Sinn der Spielidee von Tic-Tac-Toe) nicht voneinander unterscheiden.

(10 BE)

III.c **Entwickeln** Sie eine Funktion (eventuell mit Hilfsfunktionen), die den aktuellen Spielstand daraufhin überprüft, ob eine der drei Zeilen mit denselben Spielerzeichen belegt ist, so dass der jeweilige Spieler / die jeweilige Spielerin gewonnen hat. In dem Fall soll **#t** zurück gegeben werden, anderenfalls **#f**.

(5 BE)

Die in der Anlage 3 angegebene Funktion `generiere` generiert nicht (!) mit Hilfe von Tiefensuche alle Möglichkeiten, das Feld mit Zeichen zu besetzen. Sie wird mit einem leeren Feld gestartet.

III.d
- **Analysieren** Sie die Aufgabe der Funktion `generiere` in Anlage 3.
- **Geben** Sie **an**, was sie durch den Beispielaufruf in der letzten Zeile generiert.
- **Untersuchen** Sie, was in der Funktion für eine Umsetzung einer Tiefensuche fehlt.
- **Implementieren** Sie die Funktion `neuesZeichen`, die zum übergebenen Zeichen ("o" oder "x") das jeweils andere zurückgibt.

(12 BE)

Das Vorgehen von Donald Michie soll mit dem Computer so simuliert werden, dass in einer ausreichend großen Anzahl von Abfolgen von Zügen schrittweise jede so erzeugt wird, dass bei jedem Spielstand ein noch freies Feld zufällig zum Besetzen mit dem Zeichen des jeweils aktuellen Spielers ausgewählt werden soll. Mit dieser neuen Belegung soll dann jeweils der Zug fortgesetzt werden.[12]

III.e
- **Stellen** Sie **dar**, wie eine solche zufällige Auswahl eines freien Felds im Programm umgesetzt werden kann.
- **Erläutern** Sie, weshalb das beschriebene Vorgehen nicht einer realen Spielsituation entspricht.
- **Begründen** Sie, weshalb durch dieses Vorgehen das Programm trotzdem erfolgreich lernen kann, Tic-Tac-Toe zu spielen.

(10 BE)

Das Projekt für MENACE soll in einer ersten vereinfachten Version alle bereits erreichten Spielstände mit einer einfachen Bewertung entsprechend der Anzahl der Perlen verwalten. Dazu wird eine Liste `bewerteteSpielstaende` nach dem folgenden Muster verwendet:

```
'(
  ...
  ((("o" " " "x") (" " "o" " ") ("o" " " "x")) <Bewertung>)
  ...
)
```

<Bewertung> steht dabei in dieser Vereinfachung für die jeweilige Anzahl von Perlen in der Schachtel.

III.f **Implementieren** Sie die in der Anlage 4 angegebene Teilfunktion

```
(erhoeheEinen(bewertete, stand))
```

die den zu `stand` passenden bewerteten Spielstand in der Liste `bewerteteSpielstaende` der bewerteten Spielstände sucht und ihn um 1 erhöht.

(5 BE)

[12] Auf die Betrachtung der Bedeutung der Farben zur Kennzeichnung eines zu wählenden freien Nachfolgefelds wird in dieser Aufgabe zur Vereinfachung verzichtet.

Anlage zur Aufgabe MENACE (Scheme-Version)

Anlage 1 MENACE

Der britische Informatiker und KI-Forscher Donald Michie entwickelte 1960 mit MENACE („Machine Educable Noughts And Crosses Engine“) einen „Computer“ auf Basis hunderter Streichholzschachteln, der Tic-Tac-Toe lernen konnte. In den Schächtelchen, die jeweils einen möglichen Spielstand repräsentierten, waren durch verschiedenfarbige Perlen die möglichen Züge gespeichert. Je nachdem, ob ein Spiel verloren ging oder gewonnen wurde, wurden die entsprechenden Perlen entfernt oder gleichfarbige dazugelegt. Dadurch lernte das System erfolgreiche Züge und war nach einigen hundert Partien unbesiegbar.[13]

Anlage 2 Spielregeln Tic-Tac-Toe

Auf einem quadratischen, 3×3 Felder großen Spielfeld setzen die beiden Spieler abwechselnd ihr Zeichen (ein Spieler Kreuze, der andere Kreise) in ein freies Feld. Der Spieler, der als Erster drei Zeichen in eine Zeile, Spalte oder Diagonale setzen kann, gewinnt. Wenn allerdings beide Spieler optimal spielen, kann keiner gewinnen, und es kommt zu einem Unentschieden. Das heißt, alle neun Felder sind gefüllt, ohne dass ein Spieler die erforderlichen Zeichen in einer Reihe, Spalte oder Diagonalen setzen konnte.[14]

[13] https://en.wikipedia.org/wiki/Matchbox_Educable_Noughts_and_Crosses_Engine, 10.08.2025 10:49
[14] https://de.wikipedia.org/wiki/Tic-Tac-Toe, 10.08.2025 10:49

Anlage 3 Scheme-Code zur Funktion `setzeFeld`

Zu mehreren der folgenden Hilfsfunktionen ist kein Code angegeben, sondern nur eine Beschreibung, was sie machen.

```
(define
  (setzeFeld zeichen spielstand zeilenIndex spaltenIndex)
   ;;; setzt den Inhalt des Spielstands an der Position
   ;;; und gibt den neuen Spielstand zurück
  )

;;; durchsucht das Spielfeld (beginnend mit den übergebenen Indizes
;;; von hinten und unten) nach einem freien Feld
;;; und besetzt es dann mit dem übergebenen Zeichen
;;; gibt eine leere Liste zurück, wenn es kein freies Feld gibt.
(define
  (suche spielfeld zeichen zeilenIndex spaltenIndex)
  (cond
    ((negative? zeilenIndex)
     '())
    ((negative? spaltenIndex)
     (suche spielfeld zeichen (- zeilenIndex 1) 2))
    ((equal? (gibFeld spielfeld zeilenIndex spaltenIndex) " ")
     (setzeFeld zeichen spielfeld zeilenIndex spaltenIndex))
    (else
     (suche spielfeld zeichen zeilenIndex (- spaltenIndex 1)))
   )
  )
(define
  (neuesZeichen zeichen)

  ;;; Die Funktion gibt zum übergebenen Zeichen ("o" oder "x")
  ;;; das jeweils andere zurück.

  )
(define
  (gibFeld spielstand zeilenIndex spaltenIndex)
  ;;; gibt den Inhalt des Spielstands an der Position zurück
  ;;; zeilenNummer und spaltenIndex jeweils 0...2
  )

;;; Was wird generiert?
(define
  (generiere spielfeld zeichen)
  (cond
    ((equal? spielfeld '())
       #f)
    ((not
      (generiere
       (suche spielfeld zeichen 2 2)
       (neuesZeichen zeichen)))
     spielfeld)
    (else
     (generiere
      (suche spielfeld zeichen 2 2)
      (neuesZeichen zeichen)))
   )
  )
;;; Beispielaufruf
(generiere '((" "  " "  " ") (" "  " "  " ") (" "  " "  " "))  "o")
```

Anlage 4 Scheme-Code zur Funktion `erhoeheAlle`

```
;erhöht alle bewerteten Spielstände des Zugs
(define
  (erhoeheAlle bewertete zug)
  (cond
    ((null? zug)
     bewertete)
    (else
     (erhoeheAlle
      (erhoeheEinen bewertete (first zug))
      (rest zug))
     )
    )
  )

(define
  (erhoeheEinen bewertete stand)

  ;;; sucht und erhöht den zu stand passenden bewerteten Spielstand
  ;;; gibt den neu bewerteten Spielstand zurück
  ;;; siehe Aufgabenstellung

  )
```

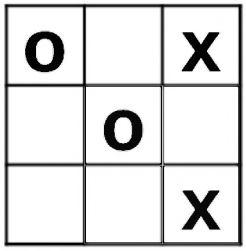
Aufgabe III: MENACE (Haskell-Version) 50 BE

Schwerpunkt: Intelligente Suchverfahren

Unter dem Namen MENACE hat Donald Michie in den 1960er Jahren eine Idee entwickelt, bei der er, noch ohne einen Computer einzusetzen, das Vorgehen moderner KI-Anwendungen vorwegnahm.

(*Text dazu und zu Tic-Tac-Toe siehe Anlagen 1 und 2*)

Für die in dem Text beschriebene Lernphase sollen in dieser Aufgabe einige Teilschritte modelliert werden.



Teilaufgaben

III.a
- **Beschreiben** Sie die folgende angegebene Datenstruktur zum Verwalten des jeweiligen Spielstands von Tic-Tac-Toe. Verwenden Sie dazu das oben dargestellte Bild als Beispiel. Dort hat Spieler 1 (Kreise) bereits zweimal gesetzt und Spieler 2 (Kreuze) ebenfalls, so dass Spieler 1 als nächster setzen kann.

```
[['x',' ','o'],[' ','o',' '],[' ',' ','x']]
```

- **Vergleichen** Sie mit einer Liste der folgenden Form:

```
['x',' ','o',' ','o',' ',' ',' ','x']
```

(8 BE)

III.b
- **Beschreiben** Sie, wie man mit Hilfe von Breitensuche alle möglichen Spielzustände nach dem Setzen der ersten drei Zeichen (also zum Beispiel Kreis, Kreuz, Kreis) bestimmen kann.
- **Bestimmen** Sie dabei auch die Anzahl der möglichen verschiedenen Abfolgen beim Setzen der drei Zeichen.
- **Begründen** Sie, dass sich viele dieser Spielzustände prinzipiell (also im Sinn der Spielidee von Tic-Tac-Toe) nicht voneinander unterscheiden.

(10 BE)

III.c **Entwickeln** Sie eine Funktion (evtl. mit Hilfsfunktionen), die den aktuellen Spielstand daraufhin überprüft, ob eine der drei Zeilen mit denselben Spielerzeichen belegt ist, so dass der jeweilige Spieler / die jeweilige Spielerin gewonnen hat. In dem Fall soll **True** zurück gegeben werden, anderenfalls **False**.

(5 BE)

Die in der Anlage 3 angegebene Funktion `generiere` generiert nicht (!) mit Hilfe von Tiefensuche alle Möglichkeiten, das Feld mit Zeichen zu besetzen. Sie wird mit einem leeren Feld gestartet.

III.d
- **Analysieren** Sie die Aufgabe der Funktion in Anlage 3.
- **Geben** Sie **an**, was sie im Beispielaufruf der letzten Zeile generiert.
- **Untersuchen** Sie, was in der Funktion für eine Umsetzung einer Tiefensuche fehlt.
- **Implementieren** Sie die Funktion `neuesZeichen`, die zum übergebenen Zeichen ("o" oder "x") das jeweils andere zurückgibt.

(12 BE)

Das Vorgehen von Donald Michie soll mit dem Computer so simuliert werden, dass in einer ausreichend großen Anzahl von Abfolgen von Zügen schrittweise jede so erzeugt wird, dass bei jedem Spielstand ein noch freies Feld zufällig zum Besetzen mit dem Zeichen des jeweils aktuellen Spielers ausgewählt werden soll. Mit dieser neuen Belegung soll dann jeweils der Zug fortgesetzt werden.[15]

III.e
- **Stellen** Sie **dar**, wie eine solche zufällige Auswahl eines freien Felds im Programm umgesetzt werden kann.
- **Erläutern** Sie, weshalb das beschriebene Vorgehen nicht einer realen Spielsituation entspricht.
- **Begründen** Sie, weshalb durch dieses Vorgehen das Programm trotzdem erfolgreich lernen kann, Tic-Tac-Toe zu spielen.

(10 BE)

Das Projekt für MENACE soll in einer ersten vereinfachten Version alle bereits erreichten Spielstände mit einer einfachen Bewertung entsprechend der Anzahl der Perlen verwalten. Dazu wird eine Liste `bewerteteSpielstaende` nach dem folgenden Muster verwendet:

```
[
    ...
    ([['x',' ','o'],[' ','o',' '],[' ',' ','x']], <Bewertung>),
    ...
]
```

<Bewertung> steht dabei in dieser Vereinfachung für die jeweilige Anzahl von Perlen in der Schachtel.

III.f **Implementieren** Sie die in der Anlage 4 angegebene Teilfunktion

`erhoeheEinen bewertete stand`

die den zu `stand` passenden bewerteten Spielstand in der Liste `bewerteteSpielstaende` der bewerteten Spielstände sucht und ihn um 1 erhöht.

(5 BE)

[15] Auf die Betrachtung der Bedeutung der Farben zur Kennzeichnung eines zu wählenden freien Nachfolgefelds wird in dieser Aufgabe zur Vereinfachung verzichtet.

Anlage zur Aufgabe MENACE (Haskell-Version)

Anlage 1 MENACE

Der britische Informatiker und KI-Forscher Donald Michie entwickelte 1960 mit MENACE („Machine Educable Noughts And Crosses Engine“) einen „Computer“ auf Basis hunderter Streichholzschachteln, der Tic-Tac-Toe lernen konnte. In den Schächtelchen, die jeweils einen möglichen Spielstand repräsentierten, waren durch verschiedenfarbige Perlen die möglichen Züge gespeichert. Je nachdem, ob ein Spiel verloren ging oder gewonnen wurde, wurden die entsprechenden Perlen entfernt oder gleichfarbige dazugelegt. Dadurch lernte das System erfolgreiche Züge und war nach einigen hundert Partien unbesiegbar.[16]

Anlage 2 Spielregeln Tic-Tac-Toe

Auf einem quadratischen, 3×3 Felder großen Spielfeld setzen die beiden Spieler abwechselnd ihr Zeichen (ein Spieler Kreuze, der andere Kreise) in ein freies Feld. Der Spieler, der als Erster drei Zeichen in eine Zeile, Spalte oder Diagonale setzen kann, gewinnt. Wenn allerdings beide Spieler optimal spielen, kann keiner gewinnen, und es kommt zu einem Unentschieden. Das heißt, alle neun Felder sind gefüllt, ohne dass ein Spieler die erforderlichen Zeichen in einer Reihe, Spalte oder Diagonalen setzen konnte.[17]

[16] https://en.wikipedia.org/wiki/Matchbox_Educable_Noughts_and_Crosses_Engine, 10.08.2025 10:49
[17] https://de.wikipedia.org/wiki/Tic-Tac-Toe, 10.08.2025 10:49

Anlage 3 Haskell-Code zur Funktion `setzeFeld`

Zu mehreren der folgenden Hilfsfunktionen ist kein Code angegeben, sondern nur eine Beschreibung, was sie machen.

```
-- setzt das Zeichen an der Position und
-- gibt den neuen Spielstand zurück
setzeFeld zeichen spielstand zeilenIndex spaltenIndex = ...


-- gibt den Inhalt des Spielstands an der Position zurück
-- zeilenNummer und spaltenIndex jeweils 0...2
gibFeld spielstand zeilenIndex spaltenIndex = ...


-- durchsucht das Spielfeld (beginnend mit den übergebenen Indizes
-- von hinten und unten) nach einem freien Feld
-- und besetzt es dann mit dem übergebenen Zeichen >>>
-- gibt eine leere Liste zurück, wenn es kein freies Feld gibt.
suche spielfeld zeichen zeilenIndex spaltenIndex
  | zeilenIndex < 0     = []
  | spaltenIndex < 0    = suche spielfeld zeichen (zeilenIndex - 1) 2
  | (gibFeld spielfeld zeilenIndex spaltenIndex) == ' '
    = setzeFeld zeichen spielfeld zeilenIndex spaltenIndex
  | otherwise           = suche spielfeld zeichen zeilenIndex (spaltenIndex
- 1)

-- Die Funktion gibt zum übergebenen Zeichen ("o" oder "x")
-- das jeweils andere zurück.
neuesZeichen zeichen  = ...

-- Was wird generiert?
generiere []          _          = []
generiere spielfeld zeichen
  | ergebnis == []              = spielfeld
  | otherwise                   = ergebnis
  where
    ergebnis = generiere (suche spielfeld zeichen 2 2) (neuesZeichen
zeichen)


--Beispielaufruf:
generiere [[' ',' ',' '],[' ',' ',' '],[' ',' ',' ']] 'o'
```

Anlage 4 Haskell-Code zur Funktion `erhoeheAlle`

```
-- erhoeht alle bewerteten Spielstände des Zugs
-- erhoeheAlle bewertete zug
erhoeheAlle bewertete []          = bewertete
erhoeheAlle bewertete (z1:rest)   = erhoeheAlle (erhoeheEinen bewertete z1) rest

-- sucht und erhöht den zu stand passenden bewerteten Spielstand
-- gibt den neu bewerteten Spielstand zurück
-- siehe Aufgabenstellung
erhoeheEinen bewertete stand = ...
```