

Smart Contracts

am Beispiel Ethereum



Inhalt

1	Was ist ein Smart Contract?	2
1.1	Contracts von heute und morgen.....	2
1.2	Smart Contract am Beispiel eines Verkaufsautomaten	3
1.3	Smart Contract Beispiel	4
1.4	Definition des Begriffs „Smart Contract“	4
1.5	Vorteile von Smart Contracts	4
2	Cola-Automat als Smart Contract.....	5
2.1	Aufgabe	5
2.2	Programmiersprache Solidity	5
2.3	Web-basierte-Entwicklungsumgebung „Remix“	6
2.4	Solidity-Sourcecode	7
2.5	Vertrag kompilieren, veröffentlichen, ansprechen.....	8
2.6	Aufgaben	9
2.8	Vetragsnutzung mit MetaMask	10
2.9	Theoriegrundlagen	10
2.10	Analyse mit Etherscan	10
2.11	Vertrag auf Sepolia veröffentlichen.....	11
2.12	Aufgaben	12
3	Test.....	13
3.1	Ethereum-Testfragen	13
3.2	Microsoft-Testfragen.....	14
3.3	EFI-Abschlussprüfung	16
4	Anhang	18
4.1	Solidity Befehlsübersicht	18

1 Was ist ein Smart Contract?

1.1 Contracts von heute und morgen



1.2 Smart Contract am Beispiel eines Verkaufsautomaten

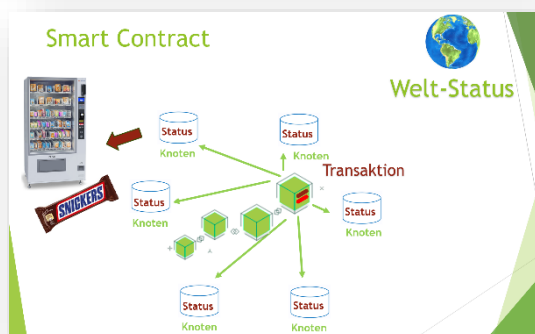


Eine einfache Metapher für einen Smart Contract ist ein Verkaufsautomat, dessen Funktionsweise mit einem Smart Contract vergleichbar ist: konkrete Eingaben garantieren vorab festgelegte Leistungen.

Der Verkaufsautomat gibt das gewünschte Produkt erst dann aus, wenn alle Anforderungen erfüllt sind. Wenn Sie beispielsweise kein Produkt auswählen oder nicht genug Geld einwerfen, gibt der Automat kein Produkt aus.



Der weltweite Abgleich einer verteilten Datenbank mit unbekannten und möglichen korrupten Teilnehmern ist eine große Herausforderung.



Die Statusänderungen eines Verkaufsvorgangs werden auf einer öffentlichen Blockchain (z.B. Ethereum) in Form von Transaktionen gespeichert und weltweit auf allen Knoten umgesetzt. Jeder kann zeitnah die Übertragung von Vermögenswerten und weitere damit verbundenen Informationen nachvollziehen.



Auf der öffentlichen Blockchain können mit Transaktionen nicht nur Coins transferiert werden, sondern auch kleine Programme mit „Wenn-Dann-Anweisungen“, sogenannte „Smart Contracts“ an die Knoten ausgeliefert werden. Die Smart Contracts werden dann über Blockchain-Transaktionen angesteuert.

1.3 Smart Contract Beispiel

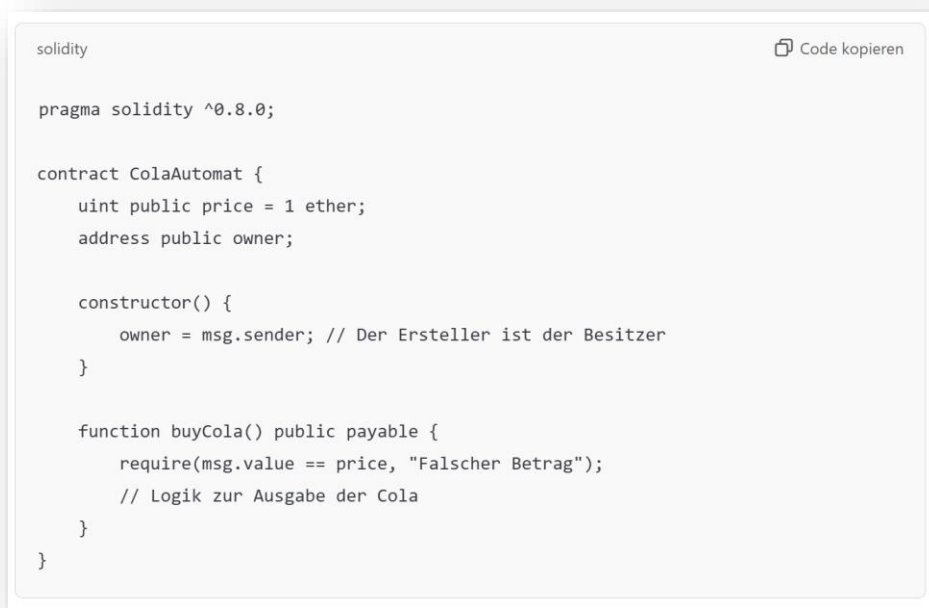
```
solidity

pragma solidity ^0.8.0;

contract ColaAutomat {
    uint public price = 1 ether;
    address public owner;

    constructor() {
        owner = msg.sender; // Der Ersteller ist der Besitzer
    }

    function buyCola() public payable {
        require(msg.value == price, "Falscher Betrag");
        // Logik zur Ausgabe der Cola
    }
}
```



1.4 Definition des Begriffs „Smart Contract“

1. Smart Contracts sind Computerprogramme, die auf der Blockchain gespeichert sind und der „Wenn dies, dann das“-Logik folgen.
2. Smart Contracts werden garantiert nach den Regeln ausgeführt, die durch ihren Code definiert sind.
3. Smart Contracts können nach der Erstellung nicht mehr geändert werden können.

Man kann sich Smart Contracts wie blockchainbasierte digitale Verträge vorstellen, die keiner menschlichen Kontrolle bedürfen, sondern auf festgelegten und manipulationssicheren Regeln basieren.

1.5 Vorteile von Smart Contracts

- **Transparenz:**
Alle Regeln sind offen einsehbar.
- **Manipulationssicherheit:**
Der Smart Contract kann nicht manipuliert oder geändert werden, weil er auf der Blockchain liegt.
- **Dezentralität:**
Kein Unternehmen oder Dritter muss das System überwachen, da der Smart Contract automatisch arbeitet.

2 Cola-Automat als Smart Contract

2.1 Aufgabe

Erstellen Sie einen **Smart Contract** in der Programmiersprache **Solidity** auf dem Testnetz **Sepolia**.

Der **Smart Contract** soll es ermöglichen, einen **Cola-Token** zu einem Preis von ca. **1,- €** mit der Währung **Ether** zu bezahlen.

Die Umsetzung dieser Aufgabenstellung wird im Folgenden erläutert.

2.2 Programmiersprache Solidity

Solidity ist eine **objektorientierte, hochspezialisierte Programmiersprache**, die primär für die Entwicklung von **Smart Contracts** auf der **Ethereum-Blockchain** und ähnlichen Plattformen konzipiert wurde. Sie wurde von **Gavin Wood**, einem der Mitbegründer von Ethereum, entwickelt und ist seit ihrer Einführung die Hauptsprache für die Erstellung dezentraler Anwendungen (DApps).

Eigenschaften von Solidity

1. **Smart Contract:** Solidity wurde speziell entwickelt, um Verträge auszuführen, die die Logik für dezentrale Anwendungen beinhalten.
2. **Syntax:** Die Syntax von Solidity ist an bekannte Sprachen wie **JavaScript**, **Python** und **C++** angelehnt. Dies erleichtert Entwicklern mit Erfahrung in diesen Sprachen den Einstieg.
3. **Turing-vollständig:** Solidity ermöglicht das Schreiben von komplexen und vielseitigen Programmen.
4. **Plattform:** Solidity wird hauptsächlich für Ethereum verwendet, ist aber auch auf anderen Ethereum-kompatiblen Blockchains (z. B. Binance Smart Chain, Polygon) einsetzbar.

Vorteile von Solidity

- Breite Unterstützung durch die Ethereum-Community.
- Starke Integration mit Blockchain-Technologien.
- Leistungsstarke Tools und Frameworks verfügbar.

Herausforderungen bei der Solidity-Programmierung

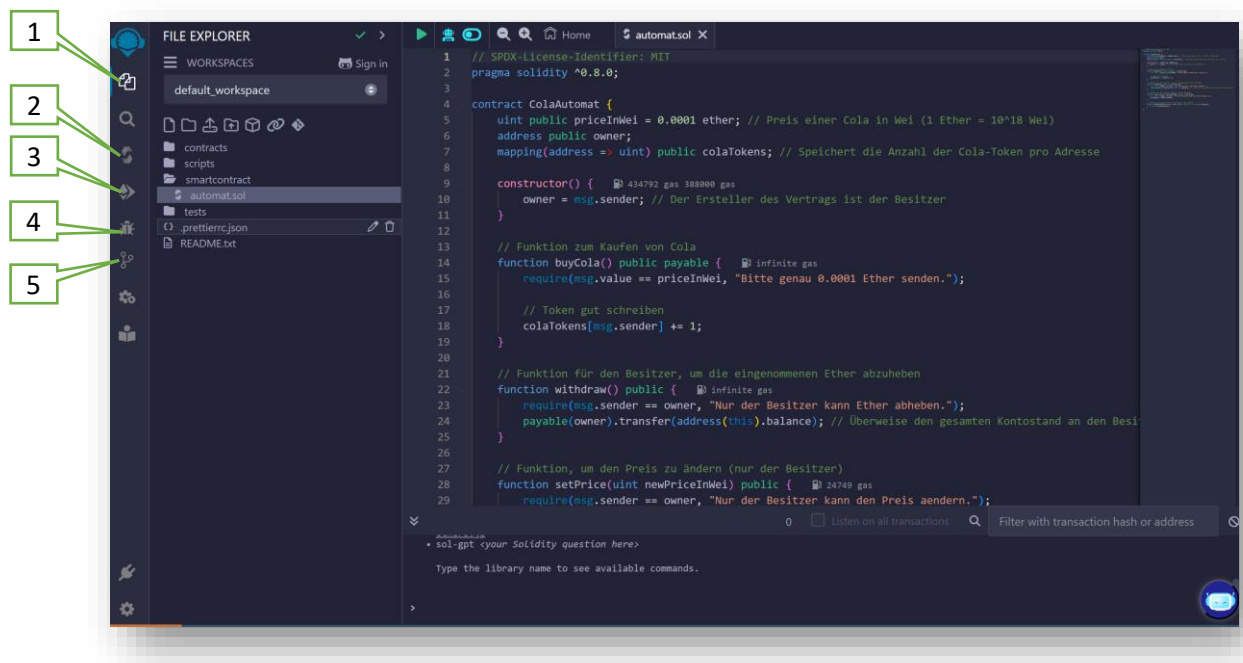
1. **Sicherheitsrisiken:**
Fehler im Code können erhebliche finanzielle Verluste verursachen.
2. **Komplexität:**
Entwickler müssen sowohl Blockchain- als auch Solidity-Kenntnisse haben.
3. **Gas-Kosten:**
Ineffizienter Code kann teure Transaktionen verursachen.

2.3 Web-basierte-Entwicklungsumgebung „Remix“

Remix ist eine **Web-basierte Entwicklungsumgebung (IDE)**, die speziell für die Erstellung, das Testen und das Bereitstellen von **Smart Contracts** auf der **Ethereum-Blockchain** entwickelt wurde. Es ermöglicht Entwicklern, schnell und einfach mit **Solidity**, der Programmiersprache für Ethereum Smart Contracts, zu arbeiten. Remix läuft vollständig im Browser und ist damit **plattformunabhängig**. Somit muss keine Software installiert werden und man kann sofort loslegen.

Die Plattform ist zu erreichen unter: <https://remix.ethereum.org/>

Der Browser merkt sich automatisch (auch ohne Account) alle Einstellungen, Ordnerstrukturen und Programme über Cookies.



1. **Code-Editor:** Ein leistungsstarker Texteditor, der speziell für Solidity entwickelt wurde. Er bietet Syntax-Highlighting, Autovervollständigung und andere Funktionen, die das Schreiben von Code erleichtern.
2. **Solidity-Compiler:** Remix enthält einen integrierten Solidity-Compiler, der den geschriebenen Code in Bytecode übersetzt, den die Ethereum-Blockchain ausführen kann.
3. **Deployment:** Sobald der Smart Contract fertig ist, kannst du ihn direkt von der Remix-Oberfläche aus auf das Ethereum-Netzwerk hochladen und in Betrieb nehmen.
4. **Testing & Debugging:** Remix ermöglicht es, Smart Contracts auf verschiedenen Testnetzwerken (wie Rinkeby oder Ganache) zu testen, bevor sie auf das Hauptnetzwerk (Mainnet) hochgeladen werden. Es bietet auch eine detaillierte Debugging-Ansicht.
5. **Git-Button:** Remix ermöglicht die Integration von Git, einem Versionskontrollsystem, direkt in die Remix-IDE. Diese Funktion ist besonders nützlich, um Smart-Contract-Projekte zu verwalten, Änderungen nachzuverfolgen und Projekte mit anderen zu teilen.

2.4 Solidity-Sourcecode

Erstellen Sie in Remix einen Ordner „**smartcontract**“ und darin die Datei „**automat.sol**“.
Übertragen Sie den folgenden Sourcecode:



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract ColaAutomat {
    uint public priceInWei = 0.0001 ether; // Preis einer Cola in Wei (1 Ether = 10^18 Wei)
    address public owner;
    mapping(address => uint) public colaTokens; // Speichert die Anzahl der Cola-Token pro Adresse

    constructor() {
        owner = msg.sender; // Der Ersteller des Vertrags ist der Besitzer
    }

    // Funktion zum Kaufen von Cola
    function buyCola() public payable {
        require(msg.value == priceInWei, "Bitte genau 0.0001 Ether senden.");

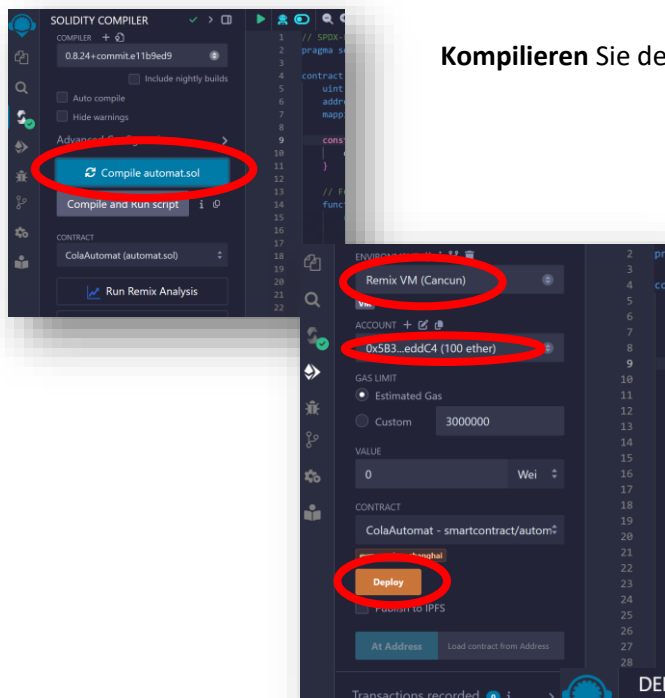
        // Token gut schreiben
        colaTokens[msg.sender] += 1;
    }

    // Funktion für den Besitzer, um die eingenommenen Ether abzuheben
    function withdraw() public {
        require(msg.sender == owner, "Nur der Besitzer kann Ether abheben.");
        payable(owner).transfer(address(this).balance); // Überweise den gesamten Kontostand an den Besitzer
    }

    // Funktion, um den Preis zu ändern (nur der Besitzer)
    function setPrice(uint newPriceInWei) public {
        require(msg.sender == owner, "Nur der Besitzer kann den Preis ändern.");
        priceInWei = newPriceInWei;
    }

    // Funktion zum Überprüfen des Cola-Tokens einer Adresse
    function getColaTokens(address user) public view returns (uint) {
        return colaTokens[user];
    }
}
```


2.5 Vertrag kompilieren, veröffentlichen, ansprechen



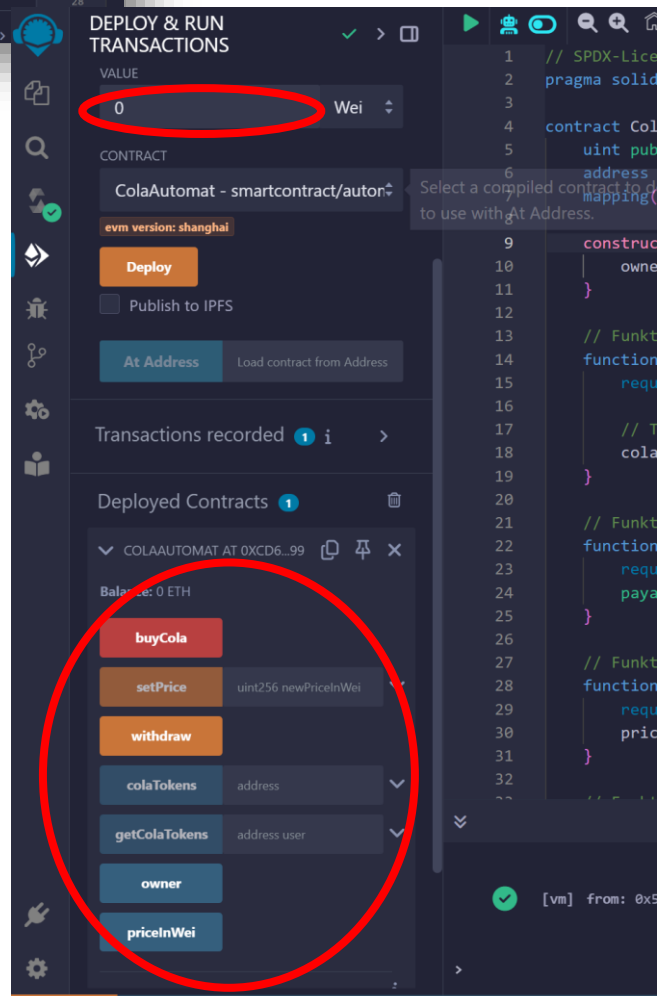
Kompilieren Sie den Sourcecode!

Veröffentlichen Sie den Smartcontract anschließend auf dem Testnetz „Remix VM“.

Über das Auswahlfeld „Account“ können Sie zwischen mehrere Konten wählen, die jeweils mit 100 Ether gefüllt sind.

Über das Eingabefeld „Value“ kann man einer Transaktion einen **Wert** mitgeben. Dieser wird benutzt, um Ether zu transferieren.

Remix analysiert den Sourcecode des Vertrages, erkennt alle ansprechbaren Elemente (Konstruktoren, Funktionen, Variablen) und generiert eine **Schnittstelle**, um entsprechende Transaktionen an den Vertrag zu senden.



2.6 Aufgaben

Aufgabe 1:

- a. Ermitteln Sie über die Schnittstelle in Remix den „Preis in Wei“.
Tragen Sie diesen Preis bei Remix als „Value“ ein und kaufen Sie einen Cola-Token!
- b. Erläutern Sie den Zusammenhang von Ether, GWei und Wei.

- c. Setzen Sie den Preis (über „SetPrice“) so, dass der Cola-Token ca. 1,- € kostet.

Preis in Wei: _____

- d. In Solidity wird der Betrag 0,0001 Ether nicht als Kommazahl, sondern als sehr großer Betrag in der Einheit Wei gespeichert. Recherchieren Sie die Begründung!

Aufgabe 2:

Analysieren Sie den Programmcode. Nutzen Sie hierfür die Solidity-Befehlsübersicht aus dem Anhang.

Erläutern Sie die folgenden Programmfunktionen in Stichpunkten.

Bereiten Sie eine entsprechende Remix-Kurzpräsentation vor.

buyCola

setPrice

uint256 newPriceInWei

withdraw

colaTokens

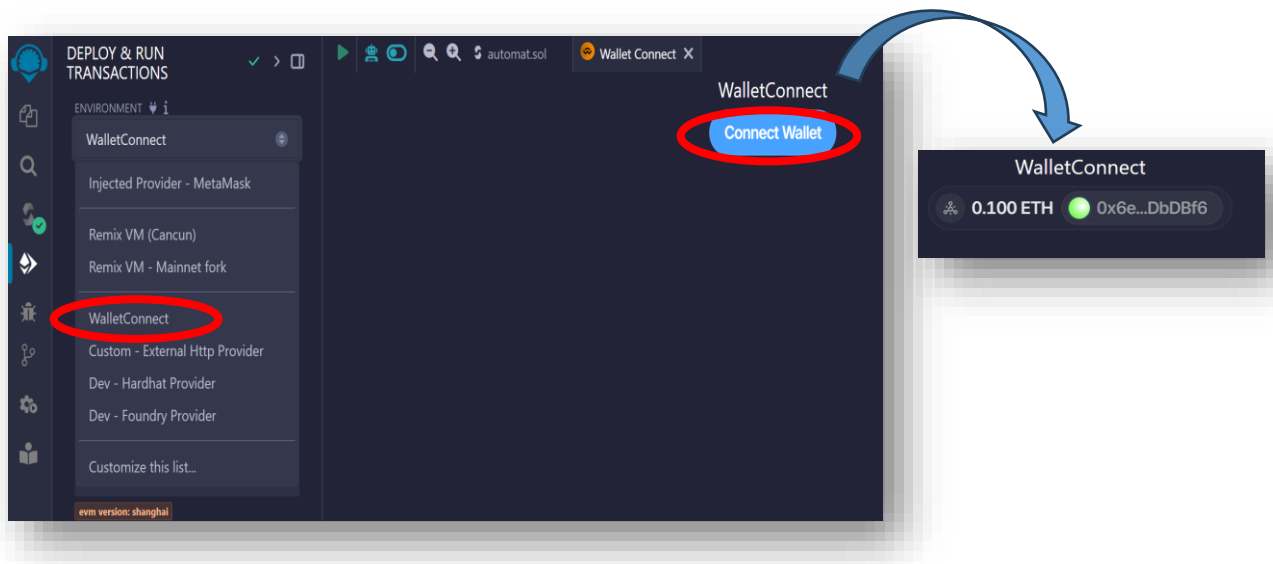
address

getColaTokens

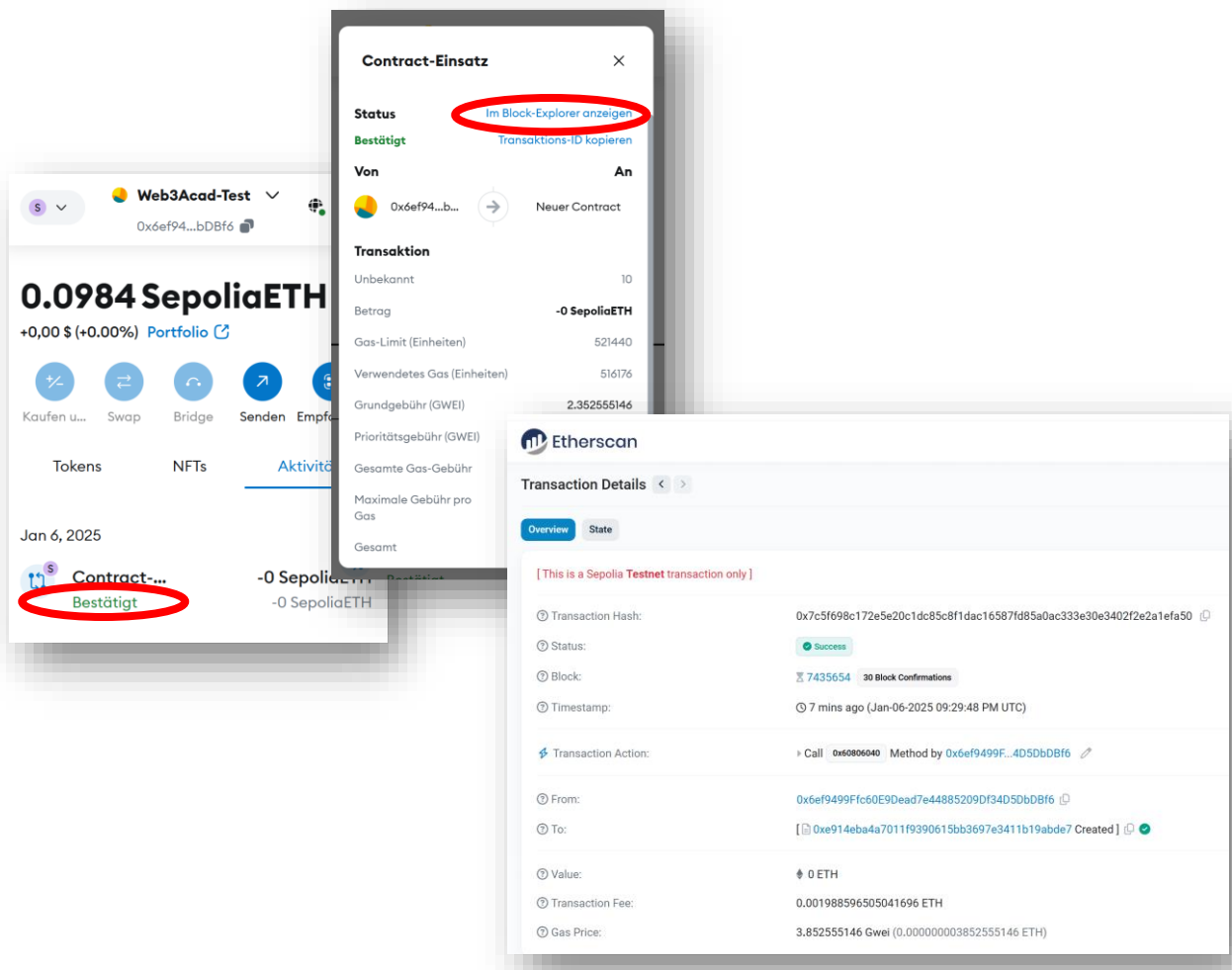
address user

2.11 Vertrag auf Sepolia veröffentlichen

Damit wir den Vertrag auf dem Sepolia-Testnetz veröffentlichen können, müssen Remix mit dem Testnetz verbunden werden. Wie man in der „Environment-Auswahlbox“ sieht, wird das Sepolia-Testnetz leider nicht direkt angeboten. Daher müssen wir erst unsere MetaMask-Wallet mit Remix verbinden und dann in MetaMask das Sepolia-Netz auswählen.



Veröffentlichen Sie nun den Vertrag auf dem Sepolia-Testnetz und analysieren Sie das Ergebnis in der MetaMask-Wallet und im Block-Explorer.



2.12 Aufgaben

Aufgabe 3:

Warum ist das Gas-Limit bei der Ausführung eines Ethereum-Smart Contracts wichtig?

Gas-Limit (Einheiten)	521440
Verwendetes Gas (Einheiten)	516176

3 Test

3.1 Ethereum-Testfragen

Beantworten Sie bitte die Testfragen zu den Ethereum-Grundlagen.

<https://ethereum.org/de/quizzes/>

Ethereum-Grundlagen

In diesem Abschnitt werden die grundlegenden Konzepte von Ethereum behandelt. So können Sie sich ein solides Basiswissen aneignen.

1. Was ist Ethereum? ✓

5 FRAGEN BEGINNER

Start

2. Was ist Ether (ETH)? ✓

4 FRAGEN BEGINNER

Start

3. Wallets ✓

4 FRAGEN BEGINNER

Start

4. Was ist Web3? ✓

5 FRAGEN BEGINNER

Start

5. Ethereum – Sicherheits- und Betrugsvermeidung ✓

5 FRAGEN BEGINNER

Start

3.2 Microsoft-Testfragen

1. Angenommen, das Kühlsystem eines Kühllastwagens fällt während der Fahrt zum Geschäft, das das Speiseeis verkaufen soll, aus. Das Speiseeis der Lieferung beginnt aufzutauen. Das Kühlsystem wird repariert, und das Speiseeis wird vor der Lieferung wieder eingefroren. Wie könnte das Geschäft, das das Speiseeis verkaufen soll, herausfinden, dass die Lieferung nicht konform ist? *

- ☐ Die Lieferung war zeitweilig nicht konform. Wenn die Kühleinheit jedoch repariert wird, nähert sich die Temperatur wieder dem Gefrierpunkt. Die letzte Transaktion vom IoT-Temperatursensor aktualisiert die Temperatur im Ledger. Die Lieferung scheint deshalb also konform zu sein.
- ☐ Die Smart-Contract-Logik legt die Lieferung als nicht konform fest, wenn die Temperatur zu hoch ist. Das Geschäft, das das Speiseeis verkaufen soll, kann überprüfen, ob die Lieferung konform ist, bevor es die Lieferung akzeptiert.
- ☐ Der Blockchainverlauf muss überwacht werden, um Aussagen über die Konformität treffen zu können. Sobald der Blockchainverlauf analysiert wurde, können Sie die Lieferung als nicht konform markieren.

2. Warum ist Blockchain unveränderlich? *

- ☐ Blockchain verwendet Hashwerte, um die Daten in der Blockchain vor Manipulationen zu schützen. Bei der Prüfung würden ungültige Blöcke erkannt werden. Fehlerhafte Blöcke würden während des Konsensprozesses ausgegeben werden. Aus diesem Grund können Blockchains in einem fehlerfreien Blockchainnetzwerk nicht geändert werden.
- ☐ Blockchainledgers sind standardmäßig für Teilnehmer schreibgeschützt. Nur Prozesse, denen Berechtigungen gewährt wurden, können Daten schreiben. Da Ledgers für Teilnehmer*innen schreibgeschützt sind, gelten sie als unveränderlich.
- ☐ Die Unveränderlichkeit hängt davon ab, wie Sie das Blockchainnetzwerk konfigurieren. Es kann so konfiguriert werden, dass es schreibgeschützt oder unveränderlich ist. Blockchain kann jedoch vollständig konfiguriert werden. Sie können Konten so konfigurieren, dass Administratoren Änderungen am Transaktionsverlauf vornehmen können. Diese Änderungen werden automatisch synchronisiert.

3. Wie ermöglichen Smart Contracts Unternehmensworkflows? *

- ☐ Bei Smart Contracts wird eine vordefinierte Logik für Unternehmensworkflows für gängige Geschäftsszenarios verwendet.
- ☐ Bei Smart Contracts handelt es sich um Clientanwendungen, die Sie schreiben und die die Blockchain aufrufen, damit diese Transaktionen ausführt und den Ledgerzustand ändert.
- ☐ Smart Contracts enthalten Zustände und programmierbare Logik. Transaktionen instanziierten Smart Contracts und führen Funktionen aus, die den Zustand ändern. Deshalb können Sie mithilfe von Smart Contracts einen Unternehmensworkflow erstellen.

4. Was enthält die erste Zeile einer Smart Contract-Quelldatei in der Regel? *

- ☐ Eine Vertragsdefinition
- ☐ Eine Pragma-Anweisung
- ☐ Eine Solidity-Version
- ☐ Ein Ereignis

5. Was sind Smart Contracts? *

- ☐ Programme, die nur Kryptowährung senden.
- ☐ Programme, die nur Token generieren und senden.
- ☐ Programme, die in einer Blockchain gespeichert werden. Sie stellen eine Vereinbarung zwischen Parteien dar, die digitale Ressourcen senden.
- ☐ Programme, die außerhalb einer Blockchain gespeichert werden. Sie müssen von Parteien signiert werden, die digitale Ressourcen senden.

6. Wie lautet ist die umfassendste Definition von „Blockchaintoken“? *

- ☐ Blockchaintoken sind eine Währung, die gehandelt werden kann.
- ☐ Blockchaintoken stellen digitale Assets dar, die einen Wert haben und gehandelt werden können.
- ☒ Blockchaintoken sind physische Token, mit denen in Blockchainnetzwerken gehandelt werden kann.
- ☐ Blockchaintoken sind eindeutige Assets, die Besitzrechte ausdrücken.

7. Welche Art von Token wird am häufigsten nur für Non-Fungible-Assets verwendet? *

- ☐ ERC20
- ☐ ERC777
- ☐ ERC1155
- ☐ ERC721

8. Worauf werden Solidity-Smart Contracts ausgeführt? *

- ☐ In der Ethereum-Blockchain
- ☐ Auf dem virtuellen Computer für Ethereum (EVM)
- ☐ Auf einem beliebigen virtuellen Computer
- ☐ Eine beliebige Sandboxumgebung

<https://learn.microsoft.com/de-de/training/paths/ethereum-blockchain-development/>

3.3 EFI-Abschlussprüfung

2. Aufgabe (25 Punkte)

- a) Sie arbeiten in der Netzleitstelle der Dunkel AG, wo Sie mit Störungen im Versorgungsnetz konfrontiert werden. Die Netzstörungen werden geloggt, damit Entstehung, Verlauf und Lösung protokolliert werden und so nachvollziehbar bleiben. Sie haben sich kürzlich mit der Blockchain-Technologie auseinandergesetzt und überlegen, ob sich diese Technologie auch zur Aufzeichnung der Netzstörungen anwenden ließe.
- aa) Sie finden im Internet einen Artikel über intelligente Verträge, sogenannte Smart Contracts, die auf einer Blockchain laufen. Sie sollen anhand des Artikelauszugs Ihren deutschsprachigen Kollegen erläutern, warum der Einsatz von Smart Contracts vorteilhaft ist.

Notieren Sie dazu zwei geeignete Argumente.

4 Punkte

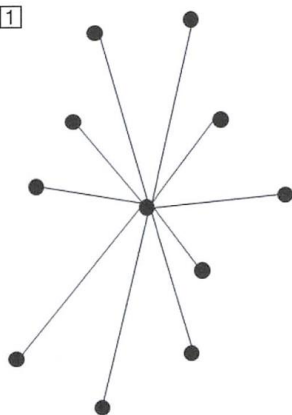
Blockchain as a so-called digital ledger is a place to record and store transactions. The first successful cryptocurrency and blockchain application was released in 2009 by Satoshi Nakamoto. Besides many other applications, blockchains provide decentralized tamper-proof open platforms to run self-executing smart contracts. Using these smart contracts, parties who haven't built trust for one another can perform transactions. Generally, a smart contract is a tiny program which is executed and validated automatically on a blockchain without a third party, when defined conditions are met. Validation means the agreement upon the correctness of the data between nodes, which is implemented in blockchains in the form of consensus mechanisms, e. g. proof-of-work. Immutability, self-execution, cost-effectiveness, accuracy, inspectability and trustlessness are in focus of smart contracts carried out on blockchains. Combining smart contracts, someone can build much complex decentralized applications that can be run on the internet without control of a centralized entity. Ethereum, EOS, and Zilliqa are examples of blockchains that are optimized for writing and executing smart contracts.

- ab) Sie haben drei Topologien zur Auswahl. Eine davon kommt der Blockchain-Topologie am nächsten.

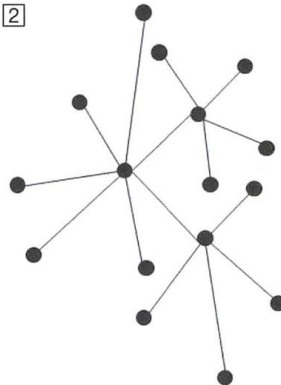
Beschreiben Sie anhand der korrekten Topologie, wie die Datenspeicherung in der Blockchain erfolgt.

2 Punkte

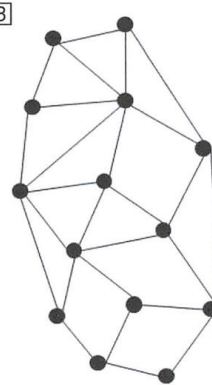
1



2



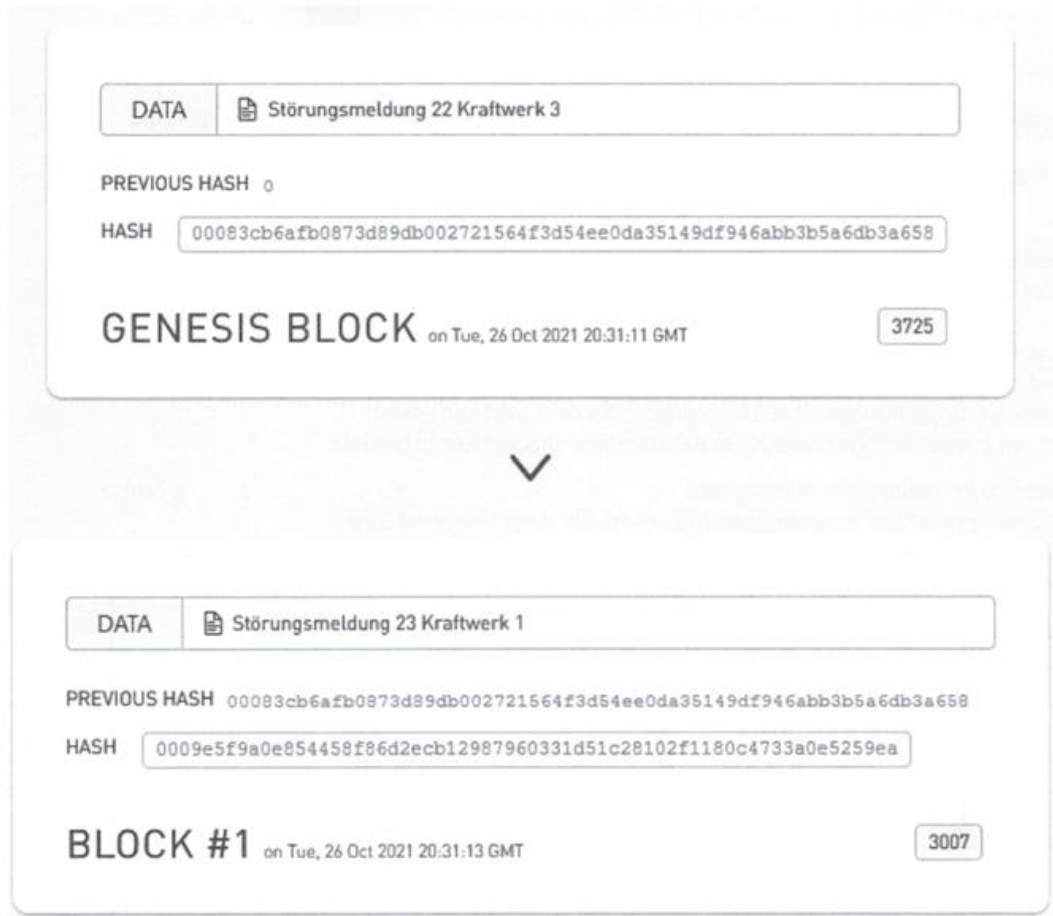
3



- ac) Die folgende Abbildung zeigt eine exemplarische Block-Struktur in einer Blockchain. Die Blöcke sind mittels einer Hash-Funktion verknüpft.

Beschreiben Sie in drei kurzen Sätzen, wie die Hash-Funktion einer Blockchain die Datenintegrität sicherstellen kann.

6 Punkte



1.

2.

3.

4 Anhang

4.1 Solidity Befehlsübersicht

Kategorie	Befehl/Schlüsselwort	Beschreibung
Versionsangabe	<code>pragma solidity ^x.x.x;</code>	Gibt die Solidity-Version an, die der Compiler verwenden soll.
Lizenzdeklaration	<code>// SPDX-License-Identifier</code>	Lizenztyp für den Code (z. B. MIT, GPL).
Vertragsdeklaration	<code>contract ContractName {}</code>	Definiert einen Smart Contract.
Variablen	<code>uint, int, bool, address</code>	Definiert Variablen. Unterstützt primitive Typen wie Ganzzahlen, Adressen usw.
	<code>mapping(KeyType => ValueType)</code>	Key-Value-Speicherstruktur.
	<code>struct StructName {}</code>	Definiert benutzerdefinierte Typen.
Zustandsvariablen	<code>storage, memory, calldata</code>	Gibt den Speicherort an (dauerhaft, temporär, oder nur zur Eingabe).
Sichtbarkeit	<code>public, private, internal, external</code>	Legt die Zugänglichkeit von Funktionen und Variablen fest.
Funktionen	<code>function name() public {}</code>	Definiert eine Funktion mit einer bestimmten Sichtbarkeit.
	<code>returns (Type)</code>	Gibt den Rückgabewert einer Funktion an.
	<code>view, pure</code>	Spezifiziert Funktionen, die nur lesen (view) oder keine Zustandsänderungen durchführen (pure).
	<code>payable</code>	Markiert Funktionen, die Ether empfangen können.
Konstruktor	<code>constructor() public {}</code>	Initialisiert den Smart Contract beim Deployment.
Modifier	<code>modifier onlyOwner {}</code>	Definiert Wiederverwendbare Bedingungen für Funktionen.
	<code>require(condition, message)</code>	Prüft Bedingungen und wirft Fehler, wenn diese nicht erfüllt sind.
	<code>assert(condition)</code>	Überprüft Zustände und löst Fehler aus (für kritische Tests).
Zugriffskontrolle	<code>onlyOwner</code>	Ein häufig verwendeter Modifier für Besitzer-Privilegien.
Ereignisse	<code>event EventName(Type value);</code>	Definiert ein Event zur Protokollierung von Daten auf der Blockchain.
	<code>emit EventName(value);</code>	Löst ein definiertes Event aus.
Ether senden/empfangen	<code>msg.value</code>	Gibt die gesendete Ether-Menge in Wei an.
	<code>address(this).balance</code>	Gibt das Ether-Guthaben des Contracts zurück.
	<code>payable(address).transfer(x)</code>	Sendet Ether an eine Adresse.
	<code>payable(address).send(x)</code>	Alternative Methode zum Senden von Ether (liefert bool).
Schleifen	<code>for, while, do-while</code>	Kontrollstrukturen für Iterationen.
Bedingungen	<code>if, else, require</code>	Kontrollstrukturen für Bedingungen.
Vererbung	<code>is</code>	Ermöglicht Vererbung von Contracts.
	<code>super</code>	Ruft eine Funktion der Elternklasse auf.
Bibliotheken	<code>library LibraryName {}</code>	Definiert eine wiederverwendbare Bibliothek.
Fallback/Receive	<code>fallback()</code>	Funktion, die aufgerufen wird, wenn keine andere Funktion passt.
	<code>receive()</code>	Funktion, die explizit Ether empfängt.
Arithmetik	<code>+, -, *, /, %</code>	Mathematische Operatoren.
	<code>SafeMath</code>	Bibliothek für sichere Berechnungen.
Typecasting	<code>uint256(x), address(y)</code>	Konvertiert Typen explizit.