

bloqs.at

Stefan Huber

Version 0.0.2

# Inhalt

Einführung .....	1
Bitcoin .....	1
Ethereum .....	2
Blockchain Anwendungen .....	4
Blockchain Anwendungsarchitektur .....	4
Szenario 1 .....	5
Einführung: Eine Sportwette .....	5
Smart Contracts .....	6
Implementierung mit Solidity .....	7
Nutzung des Smart Contracts .....	11
Anhang A: Ethereum Entwicklungsumgebung .....	12
Mist .....	12
geth .....	12
testrpc .....	12
truffle .....	12
Literatur .....	13

# Einführung

## Bitcoin

Im Jahr 2008 wurde von unbekannten Autoren mit dem Pseudonym *Satoshi Nakamoto* eine Publikation mit dem Titel "Bitcoin: A Peer-to-Peer Electronic Cash System" [Bit2008] innerhalb einer Mailingliste [Mai2008] veröffentlicht. Die Publikation beschreibt auf 9 Seiten konzeptionell ein Transfersystem von virtuellen Tokens (Bitcoins), welche über das Internet ausgetauscht werden können. Neben dieser konzeptionellen Beschreibung lieferte Satoshi Nakamoto auch eine Open Source Implementierung von Bitcoin.

Bitcoin ist als verteiltes Peer-to-Peer Netzwerk organisiert, welches keinen zentralen Dienst erfordert. Somit kann jeder Internetnutzer die Bitcoin Software installieren und Teil des Netzwerks werden. Das Herzstück von Bitcoin stellt eine verteilte Datenstruktur dar (Blockchain), welche eine Liste mit allen Transaktionen zwischen den Netzwerkteilnehmern enthält. Veränderungen in dieser Datenstruktur können nur über einen Prozess der sich Mining nennt durchgeführt werden. Als Belohnung für ihre Arbeit bekommen Mining-Knoten Bitcoin-Token ausgeschüttet. Dies ermöglicht es neue Bitcoins im System zu erschaffen.

Zusammengefasst besteht Bitcoin aus 4 Bestandteilen bzw. Schlüsselinnovationen, welche in Kombination Bitcoin ausmachen:

- Ein dezentrales Peer-to-Peer Netzwerk: Das Bitcoin Protokoll
- Ein öffentliches Transaktionsbuch: Die Bitcoin Blockchain
- Ein Bündel an Regeln für die Transaktionsverifizierung und Tokenausgabe: Die Konsensus Regeln
- Ein Mechanismus zur dezentralen Konsenermittlung über den Status der Blockchain: Proof-of-Work Algorithmus

## Bitcoin Adressen

Das Eigentum von Bitcoins wird über digitale Signaturen sichergestellt. Dabei fungieren Bitcoin Adressen als öffentlich austauschbare Schlüssel, welche weitergegeben werden können um eingehende Bitcoin Transaktion zu erhalten. Zu jeder Bitcoin Adresse gehört ein privater Schlüssel. Dieser Schlüssel kann vom Eigentümer der Bitcoin Adresse verwendet werden um Transaktionen zu signieren und damit die Zustimmung zu geben, dass von der zugehörigen Bitcoin Adresse eine Abbuchung vorgenommen werden darf. Jeder Netzwerkteilnehmer kann sich beliebig viele Bitcoin Adressen dezentral generieren. Aus Gründen der Anonymität sollte für jede Bitcoin Transaktion eine eigene Adresse generiert werden.

## Bitcoin Transaktionen

Transaktionen stellen den wichtigsten Teil innerhalb des Bitcoin Netzwerks dar. Alle anderen Bestandteile des Bitcoin Systems ermöglichen den Austausch von Transaktionen zwischen den einzelnen Netzwerkteilnehmern. Jede valide Transaktion wird von Mining-Knoten über die Propagation eines Blockes in die Blockchain aufgenommen. Diese Aufnahme in die Blockchain

führt zu einer Veränderung der Beträge hinter den Bitcoin Adressen und ermöglicht somit die Werttransfers.

## Bitcoin Mining

Der Prozess zur Generierung eines neuen gültigen Blocks und damit die Veränderung der Blockchain wird Mining genannt.

## Ethereum

Ethereum ist als verteiltes Rechnernetzwerk konzipiert, sodass jegliches Programm (Turing-Completeness) innerhalb einer eigenen isolierten Ablaufumgebung (Ethereum Virtual Machine) deterministisch und reproduzierbar ausgeführt werden kann.

Das Ethereum Protokoll ist in ständiger Weiterentwicklung. Es wurden jedoch Design Prinzipien [\[Eth2017\]](#) festgelegt, welche die zukünftige Weiterentwicklung leiten. Im folgenden werden diese Prinzipien dargelegt:

- **Einfachheit:** Das grundlegende Protokoll soll so einfach wie möglich sein, sodass ein Programmierer in der Lage sein soll die Spezifikation gänzlich zu verstehen und nötigenfalls zu implementieren. Komplexität durch Optimierungen soll nicht vermieden werden sonder in eine Mittelschicht zwischen Basisprotokoll und Anwendung platziert werden.
- **Freiheiten:** In Analogie zum Konzept der *Net Neutrality* soll auch im Ethereum Protokoll keinerlei Bevorzugung oder Benachteiligung von bestimmten Transaktionen oder Contracts geschehen können.
- **Generalisierung:** Die Konzepte im Ethereum Protokoll sollen auf dem niedrigsten Level dargestellt werden, um sie dadurch beliebig miteinander kombinierbar zu machen.
- **Ohne Features:** Als folge der Generalisierung enthält das Ethereum Protokoll keinerlei Features welchen häufige Anwendungsfälle zugrundeliegen. Jegliche Funktionen entstehen aus der Kombination bzw. Nutzung der generalisierten Konzepte.
- **Risikobereitschaft:** Ist okay ein höheres Risiko mit einer Implementierung einzugehen, wenn dies nachhaltige Vorteile bringt.

## Ethereum Konto (Account)

Ethereum Konten werden über eine 20 byte Adresse identifiziert (zB 0x829BD824B016326A401d083B33D092293333A830) und können genutzt werden um die Ethereum eigene Kryptowährung Ether über Transaktionen auszutauschen. Jedes Konto hat somit einen Kontostand, welcher in Ether geführt wird. In Ethereum können prinzipiell zwei Arten von Konten unterschieden werden:

- **Externe Konten:** Diese Konten werden von einem privaten Schlüssel kontrolliert. Über den privaten Schlüssel können Transaktionen signiert werden und somit der Kontostand verändert werden.
- **Vertragskonten:** Auf einem Vertragskonto findet sich der ausführbare Programmcode eines Smart Contract, welcher über Transaktionen aktiviert werden kann. Vertragskonten besitzen

ebenfalls einen Speicher, welcher den Zustand des Smart Contracts repräsentiert.

## Ethereum Transaktionen

Zwischen den Ethereum Konten können Nachrichten bzw. Transaktionen ausgetauscht werden. Eine Transaktion enthält primär folgende Daten:

- Die Menge an Ether, welche vom Sender zum Empfänger übertragen werden soll.
- Falls der Empfänger ein Smart Contract ist, können auch Daten übertragen werden, welche als Eingabewerte für den Programmcode gelten.
- Ein sog. Gaspreis, welcher angibt wieviel die Transaktionsausführung maximal Kosten darf.

Neben Transaktionen zwischen bestehenden Ethereum Konten, gibt es eine spezielle Transaktion, welche zur Erstellung eines neuen Vertragskontos verwendet wird. Falls der Empfänger der Transaktion die leere Adresse ("0") ist, dann wird innerhalb des Netzwerkes ein neues Vertragskonto erstellt mit den mitgeführten Daten als Programmcode. Die Adresse des Vertragskontos wird aus der Absender Adresse abgeleitet.

## Ethereum Zustandsübergänge und Codeausführung

Jede Transaktion führt bei Erfolg zu einer Zustandsänderung der Blockchain und wird innerhalb einer isolierten Instanz der Ethereum Virtual Machine ausgeführt. Jeder Schritt dieser Ausführung ist mit einer definierten Menge an GAS als Kosten bepreist. Jeder Transaktion muss daher genügend Ether als GAS mitgegeben werden, sodass die Ausführung finanziert werden kann. GAS wird somit als effizientes Mittel gegen Denial-of-Service Attacken im Ethereum Netzwerk eingesetzt. Folgendes muss für eine erfolgreiche Transaktionsausführung gelten:

- Die Transaktion muss wohl geformt und vom Absender korrekt signiert sein
- Es muss genügend Ether (für GAS und den Werttransfer) am Konto des Absenders vorhanden sein
- Es muss genügend GAS freigegeben werden um die Transaktion durchzuführen

Falls eine Transaktion fehlschlägt, werden alle Zustandsänderungen revidiert und das gesendete Ether wieder an den Absender rückgeführt. Bei einer erfolgreichen Transaktion wird die Zustandsänderung innerhalb eines Blockes gespeichert und ist nach Validierung der Miner innerhalb der Blockchain verfügbar.

## Ethereum Blockchain und Mining

Die Ethereum Blockchain ist der Bitcoin Blockchain sehr ähnlich. Jegliche Zustandsänderung eines Kontos, sei es am Kontostand oder am Speicher, muss über einen Konsens im Netzwerk geschehen. Das Ethereum Netzwerk basiert ebenfalls auf einem Verbund von vielen und weltweit verteilten Rechnern. Die Mining Knoten im Netzwerk validieren Transaktionen über den oben genannten Prozess und propagieren die Transaktion bei Erfolg innerhalb eines Blockes. Das Ethereum Netzwerk hat eine sehr schnelle Block Validierungs Zeit von ungefähr 17 Sekunden, sodass eine gültige Transaktion relativ rasch integriert wird.

# Blockchain Anwendungen

Die Blockchain Technologie wird nach Swan[\[Swa2015\]](#) als hoch disruptive Technologie eingestuft und in 3 Kategorien eingeteilt:

- **Blockchain 1.0 - Währungen:** Mit Bitcoin haben Blockchains ihren Ursprung genommen. Bitcoin oder ähnliche Altcoins (zB Litecoin, Dash oder Monero) sind sog. Kryptowährungen und können für Werttransfers über das Internet genutzt werden.
- **Blockchain 2.0 - Verträge:** In diese Kategorie würden alle Anwendungen fallen die mehrstufige Handlungsmuster umfassen, welche durch Regeln oder Wenn-Dann Beziehungen verknüpft sind. Dies könnten Crowdfunding Verträge, Initial Coin Offerings (ICOs), jegliche Formen von Wertpapieren bis hin zu Dezentralen Autonomen Organisationen (DAOs) sein.
- **Blockchain 3.0 - Dezentrale Anwendungen:** Hierin fallen ein breites Spektrum von Anwendungen, welche Einfluss auf politische Prozesse und den Rechtsstaat nehmen könnten. Diese Kategorie ist eher als Ausblick zu verstehen, folgendes könnte dies betreffen: Demokratische Prozesse und Wahlen, Ausstellung von Identitätsdokumenten, Notariat bis Hin zur Abbildung des Gesundheitswesens.

---

## Blockchain Anwendungsarchitektur

Klassische Softwareanwendungen sind nach einem Client-Server Paradigma strukturiert. Teile der Anwendung finden sich dabei im Client und andere Teile im Server. Dies resultiert in einer sog. Schichtenarchitektur (auch Multitier Architektur genannt). Dabei kann eine Anwendung innerhalb von 3 Schichten betrachtet werden:

- **Präsentationsschicht:** Bei dieser Schicht spricht man vom Frontend oder auch Client. Diese ist für die Präsentation der Anwendungsdaten an den Benutzer bzw. für die Aufnahme von Eingaben vom Benutzer verantwortlich. In Web-Anwendungen würde sich die Präsentationsschicht im Web-Browser finden.
- **Anwendungsschicht:** Innerhalb dieser Schicht findet sich die Geschäftslogik der Anwendung. Dies sind alle Regeln einer Domäne, welche die Anwendung abbildet. Gängig ist der sog. Thin Client Ansatz, in dem sich diese Schicht gänzlich am Server befindet. Eine sog. Fat Client Variante wäre ebenfalls möglich, in der sich diese Schicht im Client befindet.
- **Datenhaltungsschicht:** Alle Daten welche langfristig innerhalb der Anwendung verfügbar sein müssen, werden über diese Schicht verwaltet. Die Schicht findet sich grundsätzlich am Server, da alle Benutzer der Anwendung mit den Daten interagieren müssen.

Jede Schicht bietet zur anderen eine Kommunikationsschnittstelle, welche als Application Programming Interface (API) bezeichnet wird. Die Anwendungsschicht könnte zB eine Restful API anbieten, welche der Client nutzen kann. Die Datenschnittstelle könnte eine SQL API anbieten über diese könnte die Anwendungsschicht mit der Datenbank kommunizieren.

Blockchains bieten ähnlich wie klassische Datenbanken Kommunikationsschnittstellen. Eine Anwendung kann diese Schnittstellen nutzen um lesenden oder schreibenden Zugriff auf die Blockchain bekommt.

# Szenario 1

In diesem Kapitel soll anhand eines Beispiels das Konzept der Smart Contracts eingeführt und erklärt werden. Im ersten Abschnitt ([Einführung: Eine Sportwette](#)) wird das zugrunde liegende Beispiel einer Sportwette erklärt. Ein weiterer Abschnitt ([Smart Contracts](#)) soll weitere Details zu Smart Contracts erläutern. Gefolgt wird dies ([Implementierung mit Solidity](#)) von einer Beschreibung zur Implementierung mit der Programmiersprache Solidity. Abschließend wird erörtert wie Smart Contracts genutzt werden können ([Nutzung des Smart Contracts](#)).

## Einführung: Eine Sportwette

Am Beispiel einer Sportwette sollen die Funktionsweise und die Eigenschaften eines Smart Contracts aufgezeigt werden. Dazu wird die Sportwette auf das Ergebnis eines Fußballspiels betrachtet. Ein Fußballspiel zwischen zwei Mannschaften A und B kann generell drei Ergebnisse bzw. Endzustände aufweisen:

- Mannschaft A gewinnt und Mannschaft B verliert
- Mannschaft B gewinnt und Mannschaft A verliert
- das Spiel geht unentschieden aus

Um nun eine Wette auf eines der möglichen Ergebnisse abzugeben, würde normalerweise ein sog. Buchmacher konsultiert. Dieser würde Wetten von unterschiedlichen Teilnehmern, mit unterschiedlichen Wetteinsätzen und natürlich unterschiedlichen Tipps entgegennehmen. Dies sollte klarerweise alles vor dem Start eines Fußballspiels geschehen. Aus den gesamten abgegebenen Wetten kann dann die sog. Gewinnquote für die jeweiligen Ergebnisse berechnet werden. Diese Quote gibt an mit welchem Faktor der Wetteinsatz im Falle eines richtigen Tipps multipliziert wird. Aus der Quote errechnet sich somit der mögliche Gewinn für die Wettteilnehmer. In Tabelle 1 wird ein Beispiel eines möglichen Wettverlaufs dargestellt. In diesem vereinfachten Beispiel ist davon auszugehen, dass keine Gebühren für den Buchmacher bzw. keine Steuern erhoben werden.

Tipp	Ergebnis	kumulierte Einsätze	Prozent	Gewinnquote
Tipp 1	Mannschaft A gewinnt	€ 20.000	25%	4
Tipp 2	Mannschaft B gewinnt	€ 50.000	62,5%	1,6
Tipp 3	unentschieden	€ 10.000	12,5%	8
	gesamt	€ 80.000	100%	

Die Wettquote berechnet sich somit konkret über den gesamten Wetteinsatz dividiert durch die kumulierten Wetteinsätze aller Teilnehmer für den jeweiligen Tipp. Die Gewinnquote bedeutet somit für einen Wettteilnehmer, dass für jeden Euro der investiert wird das 4-, 1,6- bzw. 8-fache als Gewinn verbucht werden kann. An den Beispielzahlen ist zu erkennen, dass eine Gewinnquote immer größer als 1 sein muss, andernfalls würde die Teilnahme an einer Wette keinen Sinn machen. Zusätzlich ist zu sehen, dass der Tipp mit dem kleinsten kumulierten Wetteinsatz die höchste Gewinnchance bietet. Eine Wette auf das Ergebnis mit der höchsten Gewinnchance ist oft auch mit dem größten Verlustrisiko verbunden.



Wetten könnten auch als Vorhersagen betrachtet werden und die kummulierten Wetteinsätze als Eintrittswahrscheinlichkeit gedeutet werden. Im Beispiel der Fußballwette könnte dies so interpretiert werden, dass mit einer Wahrscheinlichkeit von 12,5% ein Unentschieden gespielt wird. Blockchainstartups wie **Augur** [4: Augur: <https://augur.net>], **Gnosis** [5: Gnosis: <https://gnosis.pm/>] oder **Stox** [6: Stox: <https://www.stox.com>] machen sich genau diesen Umstand zu nutze und bietet Vorhersagen der "kollektiven Intelligenz" als Dienstleistung an. Dies ist auch unter den Bezeichnungen **crowd sourcing** oder **prediction markets** bekannt.

## Smart Contracts

Üblicherweise würde eine Sportwette wie oben beschrieben von einem Buchmacher abgewickelt. Dieser Buchmacher würde sich um die ordnungsmäßige Abwicklung der Wette kümmern. Je nach Vertrauen gegenüber dem Buchmacher, würden Wettteilnehmer dort Wetten abschließen oder auch nicht. Je nach Einsicht der Wettteilnehmer könnte ein Buchmacher Gewinnquoten zu seinen Gunsten manipulieren oder hohe Gebühren für Gewinne erheben.

Die ordnungsmäßige bzw. korrekte Abwicklung einer Sportwette kann jedoch formal wie in einem Algorithmus bzw. einem Computerprogramm beschrieben werden. Diesen Umstand haben sich bereits etliche online Sportwettanbieter zu nutze gemacht. Anstelle einer vertrauenswürdigen Person, welche als Buchmacher Wetten entgegennimmt, kann dies auch über eine Software Anwendung geschehen. Hierbei wird jedoch nur die Abwicklung an sich digitalisiert, eine Manipulation ist dabei noch nicht ausgeschlossen. Gemeinhin ist es so, dass Software Anwendungen zentral verwaltet sind und es ist nicht möglich den Programmcode einzusehen. Es wäre einem Wettteilnehmer also nicht möglich zu prüfen, ob die Software des Sportwettanbieters so agiert wie angepriesen. Es könnte durchaus passieren, dass auch nach einer gewonnen Wette, der Wetteinsatz nicht ordnungsgemäß ausbezahlt wird.

Ein Smart Contract hingegen ist ähnlich einem Computerprogramm, das nachvollziehbar genau das macht, was beschrieben ist und dies unmanipulierbar durchführt. Im Englischen wird dies auch mit dem Ausspruch "Code is Law" bekräftigt. Dies deutet darauf hin, dass so wie es im Quelltext des Smart Contracts beschrieben ist, so gilt es auch (ähnlich wie ein Gesetz). Konzeptionell ist ein Smart Contract einem Computerprogramm sehr ähnlich, jedoch gibt es einige unterscheidende Merkmale:

- Smart Contracts sind autonom in ihrer Ausführung. Falls ein Smart Contract einmal im Netzwerk bereitgestellt ist, steht dieser dort für immer bereit. Die programmierten Regeln sind somit gültig und können nicht mehr abgeschaltet oder verändert werden.
- Die Kommunikation mit Smart Contracts wird über Transaktionen realisiert. Dabei kann eine Transaktion sowohl Daten als auch Werte, abgebildet als virtueller Token, enthalten. Mit der Transaktion wird dem Smart Contract die Hoheit über die übermittelten Werte übertragen.
- Für die Ausführung eines Smart Contracts muss bezahlt werden. Die Mining-Knoten, welche das Netzwerk betreiben, bekommen eine Gebühr (Mining Fee) als Entschädigung für ihre Arbeitsleistung. Jede Transaktion, welche mit dem Smart Contract interagiert, muss auch genügend Mittel bereitstellen, damit die programmierten Regeln ausgeführt werden.



Konkret ist die Ausführung des Smart Contracts in Abschnitte gegliedert. Jeder Abschnitt wird durch eine Transaktion bzw. durch ein Ereignis gestartet. Am Beispiel der Sportwette können 3 Abschnitte identifiziert werden:

- Abschnitt 1: Innerhalb dieses Abschnitts können Wettteilnehmer ihre Wetten platzieren. Diese Phase würde solange andauern, bis das Fußballspiel gestartet wird. Nach diesem Ereignis können keine Wetten mehr platziert werden.
- Abschnitt 2: Dieser Abschnitt würde während des Fußballspiels ablaufen. Der Smart Contract wäre im sog. Leerlauf und würde nur auf das Ereignis "Ende des Fußballspiels" warten.
- Abschnitt 3: Der letzte Abschnitt, nachdem das Spielergebnis feststeht, wäre die Auszahlung. Die Wettteilnehmer, welche die Wette zu ihren Gunsten platziert haben, können sich nun ihre Gewinne auszahlen.

## Implementierung mit Solidity

Der Solidity Programmcode ist innerhalb einem Verzeichnisses auf github [7: Szenario 1 auf github: <https://github.com/getbloqs/scenario01>] hinterlegt und einsehbar. In diesem Abschnitt werden einige Ausschnitte des Programmcodes angeführt und erläutert. Dies soll dazu dienen den Smart Contract zu dokumentieren und wesentliche Konzepte der Programmiersprache Solidity einzuführen.

### Listing 1

```
contract SportsBet {  
}
```

Jeder Smart Contract wird wie oben veranschaulicht (Listing 1) über das Schlüsselwort `contract` deklariert. Dies hat eine starke Ähnlichkeit zur Deklaration von Klassen in Objektorientierten Programmiersprachen (OOP). Im allgemeinen gibt es eine Vielzahl Ähnlichkeiten mit OOP, sodass für Softwareentwickler das Erlernen von Solidity eine geringe Lernkurve aufweist.

### Listing 2

```
contract SportsBet is Owned {  
}
```

Solidity unterstützt auch das Konzept der Vererbung (Listing 2). Dies ist hilfreich um die Wiederverwendbarkeit von Programmcode zu erhöhen. Über das Schlüsselwort `is` erbt `SportsBet` von `Owned`. Alle Funktionen, welche in `Owned` definiert sind, sind somit auch Teil von `SportsBet`. Neben der einfachen Vererbung werden auch `Interfaces` bzw. `abstrakte Methoden` unterstützt, dies wird in einem späteren Abschnitt noch thematisiert.

```

contract SportsBet is Owned {

    // unique identifier of sports game
    string public game;

    function SportsBet(string _game) {
        game = _game;
    }
}

```

Eine wesentliche und wichtige Eigenschaft von Smart Contracts ist es Zustände zu speichern. Dazu können Attribute bzw. Zustandsvariablen deklariert werden. In Listing 3 wird dazu eine Variable mit Namen `game` deklariert. Diese Variable hat den Datentyp `string` und kann Zeichenketten enthalten. Solidity unterstützen unterschiedlichste Datentypen, eine umfassende Liste ist der Dokumentation [8: Liste aller Solidity Datentypen: <http://solidity.readthedocs.io/en/latest/types.html>] zu entnehmen. Mit der Angabe `public` wird die Sichtbarkeit der Variable für andere Smart Contracts angegeben. Somit kann die Variable `game` von anderen Smart Contracts abgefragt werden. Neben `public` gibt es weitere Sichtbarkeitsdeklarationen, welche ebenfalls für Funktionen gelten und weiter unten eingeführt werden.



#### *Sichtbarkeit von Zuständen*

Alle Zustände, welche innerhalb eines Smart Contracts hinterlegt werden, sind grundsätzlich über die Blockchain öffentlich einsehbar. Die Sichtbarkeit einer Variable innerhalb eines Smart Contracts (`public`, `private`, `internal` oder `external`) bezieht sich dabei nur auf die programmatischen Zugriffsmöglichkeiten anderer Smart Contracts. Es gibt keine Möglichkeit Daten innerhalb einer Blockchain "nicht-öffentlich" zu speichern. Somit sind alle Daten die von einem Smart Contract zur Bearbeitung benötigt werden öffentlich.

Ein Smart Contract kann des Weiteren über einen sog. Konstruktor verfügen. Listing 3 enthält ebenfalls einen Konstruktor. Dieser ist generell nichts anderes als eine normale Funktion, mit dem Unterschied, dass der Funktionsname mit dem Namen des Smart Contracts übereinstimmen muss. Eine Funktionsdeklaration wird über das Schlüsselwort `function` durchgeführt. Die Funktion kann Übergabeparameter definieren, welche zur Konstruktion des Smart Contracts mitgegeben werden müssen.

#### Listing 4

```
contract SportsBet is Owned {

    struct Bet {
        uint tip;
        uint amount;
    }

    mapping (address => Bet) bets;

    function bet(uint tip) public payable {
        if (tip < 1) {
            tip = 1;
        } else if (tip > 3) {
            tip = 3;
        }

        if (bets[msg.sender].tip == 0) {
            bets[msg.sender].tip = tip;
        }
        bets[msg.sender].amount += msg.value;
    }

}
```

Neben einfachen Datentypen wie in [Listing 3](#) die Variable `game` können auch komplexere Datentypen selbst definiert werden. Um eine Wette zu repräsentieren wird in [Listing 4](#) der komplexe Datentyp `Bet` eingeführt. Dieser enthält einen Zahlenwert `tip` für den Wettipp und einen weiteren Zahlenwert `amount` für die Höhe des Einsatzes deklariert. `uint` deklariert dabei einen sog. unsigned Integer, also einen Zahlenwert der nur positiv sein kann. Der komplexe Datentyp wird über das Schlüsselwort `struct` deklariert.

Wie bereits erwähnt besitzen Smart Contracts Funktionen. In [Listing 4](#) werden eine Funktionen des Smart Contracts `SportsBet` implementiert. Generell stellen Funktionen (je nach Sichtbarkeit) die Schnittstelle des Smart Contracts nach Außen dar. Diese Schnittstelle wird über die sog. Signatur der Funktion definiert. Die Signatur setzt sich aus unterschiedlichen Bestandteilen zusammen:

- Der Name der Funktion
- Den spezifizierten Übergabeparametern
- Den Rückgabewerten, falls diese definiert sind
- Sichtbarkeits- bzw. sonstigen Modifikatoren

Neben der Signatur, welche auch als Funktionskopf bezeichnet werden kann, gibt es einen Funktionskörper. Der Funktionskörper wird von zwei geschwungenen Klammern umschlossen `{ }`. Innerhalb dieses Körpers wird die Logik der Funktion implementiert. Dazu werden unterschiedliche Konstrukte der Programmierung angewandt. In der Funktion `bet` aus [Listing 4](#) werden sog. Kontrollstrukturen eingesetzt, um den übergebenen Parameter zu überprüfen. Die

verfügbaren Kontrollstrukturen in Solidity können in der Dokumentation [9: Kontrollstrukturen in Solidity: <https://solidity.readthedocs.io/en/latest/control-structures.html#control-structures>] eingesehen werden. Falls die Signatur Rückgabewerte definiert, müssen diese über das Schlüsselwort `return` übergeben werden. Dieses Schlüsselwort beendet ebenfalls die Ausführung der Funktion.

#### *Sichtbarkeitsmodifikatoren*

Funktionen und Zustandsvariablen besitzen eine Sichtbarkeit innerhalb des Smart Contracts bzw. nach Außen zum Netzwerk. Solidity bietet 4 verschiedene Sichtbarkeitsmodifikatoren.



- **external**: Externe Funktionen bilden u.a. die Schnittstelle nach Außen eines Smart Contracts. Externe Funktionen können nur über Transaktionen, von anderen Smart Contracts oder über einen Message Call aufgerufen werden. Ein interner Aufruf der Funktion ist nur über `this` möglich. Zustandsvariablen können nicht als **external** deklariert werden.
- **public**: Funktionen und Zustandsvariablen können intern oder extern aufgerufen werden. Für Zustandsvariablen, welche als **public** deklariert wurden, wird automatisch eine sog. Getter-Funktion erzeugt.
- **internal**: Ein Zugriff auf **internal** Funktionen oder Zustandsvariablen ist nur vom deklarierten oder vererbten Smart Contract möglich.
- **private**: Zustandsvariablen bzw. Funktionen dieser Sichtbarkeit sind nur innerhalb des Smart Contracts ansprechbar.

Die Defaultsichtbarkeit von Funktionen ist **public** und die von Zustandsvariablen ist **internal**.

**Listing 4** enthält auch ein **mapping**. Mappings sind sehr wichtige und effiziente Strukturen zur Speicherung von Zuständen in Smart Contracts. In anderen Programmiersprachen werden Mappings auch assoziativ Speicher oder Hashtabellen genannt. Ein mapping ist demnach eine Datenstruktur um Werte anhand einen Schlüssels zu speichern bzw. abzufragen. Das deklarierte **mapping** in **Listing 4** speichert Wetten (**Bet**) anhand der Adresse (**address**) des Wetteteilnehmers.

Die Funktion **bet** in **Listing 4** speichert den übergebenen Parameter **tip** als Wette (**Bet**) innerhalb des **mapping** (**bets**). Die Funktion prüft zuerst ob ein valider Tipp (Tipp 1, Tipp 2 oder Tipp 3) abgegeben wurde. Dannach wird überprüft ob der Absender der Transaktion und somit der Aufrufer der Funktion **bet** bereits eine Wette abgegeben hat. Der Absender der Transaktion wird über die spezielle Variable **msg.sender** abgefragt. Falls noch kein Tipp abgegeben wurde, wird dieser für den Absender gesetzt. Dazu wird der Tip über das **mapping** mit dem Schlüssel **msg.sender** (**msg.sender** ist vom Datentyp **address**) in der Wette (**Bet**) gesetzt. In jedem Fall wird der Wetteinsatz erhöht. Dazu wird die Höhe des gesendeten Ethers zum Wetteinsatz (**amount**) addiert.

Die Transaktion, welche die Funktion des Smart Contracts aufruft, sendet Ether als Wetteinsatz mit. Dieses Ether wird dem Smart Contract bereitgestellt und dieser verfügt nun darüber.



### *Spezielle Variablen: Block bzw. Transaktions Eigenschaften*

Funktionen haben Zugriff auf spezielle Variablen, welche wichtige Informationen über den aktuellen Ausführungskontext enthalten. So kann zB über die Variable `msg` auf den Absender der Transaktion (`msg.sender`) den gesendeten Betrag an Ether (`msg.value`) oder das noch verfügbare Gas (`msg.gas`) zugegriffen werden. Alle verfügbaren bzw. speziellen Variablen sind in der Dokumentation [11: Spezielle Variablen in Solidity: <https://solidity.readthedocs.io/en/latest/units-and-global-variables.html#special-variables-and-functions>] einsehbar.

## Nutzung des Smart Contracts

Der Smart Contract wurde mit der Programmiersprache Solidity entwickelt, welche in einen Ethereum Smart Contract kompiliert werden kann. Das Ergebnis dieser Kompilierung ist letztlich nichts anderes als Bytecode, welcher auf der Ethereum Virtual Machine (EVM) ausgeführt werden kann.

Der kompilierte Smart Contract kann über eine Ethereum Wallet im Netzwerk bereitgestellt werden. Dies erfordert eine Transaktion an eine leere Adresse, welche als Nachricht den kompilierten Smart Contract enthält. Für diese Bereitstellung muss der Transaktion natürlich genügend Gas als Gebühr mitgegeben werden. Diese Bereitstellung führt ebenfalls den Konstruktor des Smart Contracts aus, das bereitgestellte Gas muss auch dafür reichen.

Nachdem der Smart Contract bereitgestellt wurde, kann über Transaktionen mit seiner öffentlichen Schnittstelle (`external` oder `public` Funktionen) kommuniziert werden. Der Smart Contract besitzt eine eindeutige Adresse und kann darüber identifiziert und angesprochen werden. Nutzer können Transaktionen an diese Adresse senden. Innerhalb der Nachricht können die gewünschten Funktionen bzw. Funktionsparameter angegeben werden. Am Beispiel der Sportwette würde innerhalb der Nachricht ein Tipp für das Ergebnis angegeben und die Funktion `bet` aufgerufen.

# Anhang A: Ethereum Entwicklungsumgebung

Mist

geth

testrpc

Smart Contracts, welche für die Ethereum Blockchain entwickelt werden, sollten auch entsprechend getestet werden. Erst dannach sollten die Smart Contracts im Livenet bereitgestellt werden. Neben dem Livenet gibt es für Ethereum auch reale Testnetzwerke, welche von der Ethereum Community betrieben werden. Um Smart Contracts am eigenen Rechner zu entwickeln und testen, kann auf ein simuliertes Testnetzwerk zurückgegriffen werden. Die node.js Anwendung **testrpc** stellt ein solches simuliertes Netzwerk bereit.

Der Quellcode und etwas Dokumentation zu **testrpc** findet sich auf github [12: Link zu testrpc: <https://github.com/ethereumjs/testrpc>]. Die Anwendung kann mithilfe des **node package managers** installiert werden. Dazu muss folgender Befehl ausgeführt werden.

```
$ npm install -g ethereumjs-testrpc
```

Nach Installation kann das simulierte Netzwerk über die Kommandozeile gestartet werden. Defaultmäßig wird das Netzwerk auf Port **8545** ausgeführt. Dies kann jedoch über Optionen an den Startbefehl geändert werden. Details zu den unterschiedlichen Optionen ist der Dokumentation zu entnehmen.

truffle

Truffle [13: Link zu truffle: <http://truffleframework.com>] ist ein Kommandozeilenprogramm welches die Entwicklung von Smart Contract Projekten mit Solidity stark vereinfacht und erleichtert. Truffle stellt folgende Funktionen bereit:

...

Truffle ist eine node.js Anwendung und kann über den **node package manager** installiert werden. Dazu muss folgender Befehl ausgeführt werden:

```
$ npm install -g truffle
```

# Literatur

- [Bit2008] Bitcoin: A Peer-to-Peer Electronic Cash System, Satoshi Nakamoto; Link: <http://bitcoin.com/bitcoin.pdf>
- [Swa2015] Melanie Swan. Blockchain
- [Eth2017] Design Rationale, Ethereum; Link: <https://github.com/ethereum/wiki/wiki/Design-Rationale>
- [Mai2008] Mail Archive Cypherpunk Mailingliste; Link: <http://www.mail-archive.com/cryptography@metzdowd.com/msg09959.html>