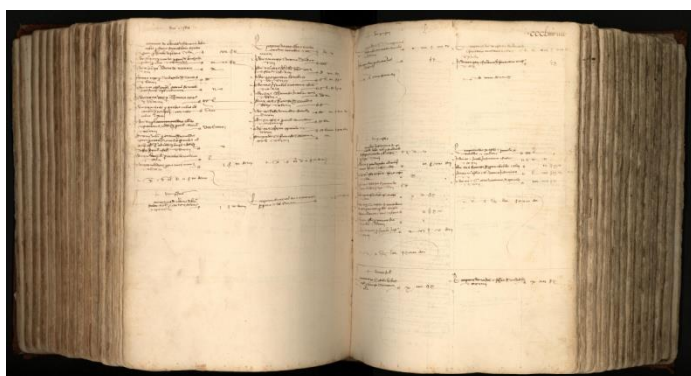


Blockchain (catena di blocchi)

1. Blockchain
2. Transazione
3. Crittografia
4. Hashing
5. Blockchain come tipo di database
6. Piccola Blockchain in Python
7. Mining
8. WeChat

Cos'è la Blockchain



Immagina un **registro digitale**, come un grande libro mastro, che è condiviso tra molte persone (o computer) in una rete. Ogni volta che avviene una **transazione o viene registrato un dato**, viene aggiunto un **"blocco" di informazioni** a questo registro. Una volta che un blocco viene aggiunto, è estremamente difficile modificarlo o cancellarlo, **perché ogni blocco è collegato crittograficamente al blocco precedente**. Questa catena di blocchi (da cui il nome "blockchain") è ciò che rende il sistema sicuro e trasparente. Ecco le caratteristiche chiave:

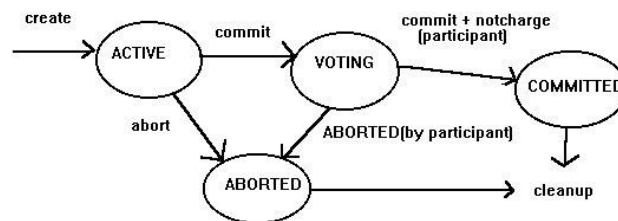
- **Decentralizzazione:** Invece di essere controllata da una singola entità (come una banca o un governo), la blockchain è distribuita su molti computer. **Una copia dell'intera blockchain è mantenuta su una rete di numerosi computer, spesso chiamati nodi**, sparsi in diverse posizioni geografiche. Questo la rende più resistente alla censura e ai guasti.
- **Trasparenza:** Tutte le transazioni (o i dati registrati) sono visibili a tutti i partecipanti della rete. Anche se l'identità delle persone coinvolte può essere pseudonima, le operazioni sono pubbliche.
- **Immutabilità:** Una volta che un blocco viene aggiunto alla catena, è quasi impossibile modificarlo retroattivamente senza il consenso di tutti i partecipanti alla rete. Questo è garantito dalla crittografia.
- **Sicurezza:** La crittografia assicura l'integrità dei dati e la validità delle transazioni. Ogni blocco contiene un "hash" (una sorta di impronta digitale unica) del blocco precedente, creando un legame indissolubile.

Esempi di Utilizzo della Blockchain:

- **Criptovalute (come Bitcoin ed Ethereum):** Questo è l'uso più noto. La blockchain registra tutte le transazioni di criptovaluta in modo sicuro e trasparente.

- **Tracciabilità della Supply Chain:** Le aziende possono utilizzare la blockchain per tracciare l'origine e il percorso dei prodotti, garantendo l'autenticità e la provenienza (ad esempio, per alimenti, farmaci, beni di lusso).
- **Votazioni Elettroniche:** La blockchain potrebbe essere utilizzata per creare sistemi di voto online più sicuri e trasparenti, riducendo il rischio di frodi.
- **Gestione dell'Identità Digitale:** La blockchain può fornire un modo sicuro e controllato dagli utenti per gestire e condividere la propria identità digitale.
- **Smart Contracts:** Sulla blockchain di Ethereum, ad esempio, è possibile creare "contratti intelligenti" che si eseguono automaticamente al verificarsi di determinate condizioni, senza la necessità di intermediari.
- **Registri Fondiari:** La blockchain può essere utilizzata per creare registri immobiliari immutabili e trasparenti, semplificando le transazioni e riducendo le dispute.

Transazione



Una **transazione** in un database è una **sequenza di una o più operazioni** (come letture, scritture, modifiche, eliminazioni) che vengono trattate come una singola unità logica di lavoro. Pensa a una transazione come a un "tutto o niente": o tutte le operazioni all'interno della transazione vengono completate con successo e le modifiche vengono rese permanenti nel database (questa azione è chiamata **commit**), oppure, se qualcosa va storto durante l'esecuzione, tutte le modifiche vengono annullate e il database torna allo stato precedente all'inizio della transazione (questa azione è chiamata **rollback**).

Esempio 1: Trasferimento di denaro da un conto bancario all'altro

Immagina di dover trasferire 100€ dal tuo conto A al conto B. Questa operazione può essere vista come una transazione composta da due passaggi:

1. **Decrementare** il saldo del conto A di 100€.
2. **Incrementare** il saldo del conto B di 100€.

È fondamentale che entrambe queste operazioni avvengano con successo. Se, per esempio, il sistema dovesse bloccarsi dopo aver decrementato il conto A ma prima di incrementare il conto B, i dati del database sarebbero inconsistenti (avresti 100€ in meno ma non sarebbero finiti da nessuna parte). Una transazione garantisce che o entrambi i passaggi vengano completati (il trasferimento avviene con successo) o nessuno dei due venga eseguito (il database rimane nello stato iniziale).

Esempio 2: Ordine di un prodotto in un negozio online

Quando effettui un ordine online, diverse operazioni devono avvenire in sequenza:

1. **Verificare** la disponibilità degli articoli richiesti.
2. **Aggiornare** l'inventario, diminuendo la quantità degli articoli ordinati.
3. **Creare** un nuovo record dell'ordine nel database.
4. **Registrare** il pagamento.

Crittografia



La **crittografia** è l'arte e la scienza di **trasformare informazioni (testo in chiaro) in una forma incomprensibile (testo cifrato)**, in modo che solo le persone autorizzate (che possiedono la "chiave" per decifrare) possano riportarle alla loro forma originale. L'obiettivo principale della crittografia è garantire la **confidenzialità**, l'**integrità**, l'**autenticità** e il **non ripudio** delle informazioni.

- **Confidenzialità:** Assicurare che solo i destinatari previsti possano leggere il messaggio.
- **Integrità:** Garantire che il messaggio non sia stato alterato durante la trasmissione o l'archiviazione.
- **Autenticità:** Verificare l'identità del mittente del messaggio.
- **Non ripudio:** Impedire al mittente di negare di aver inviato il messaggio.

La crittografia utilizza **algoritmi** (insiemi di regole ben definite) e **chiavi** (informazioni segrete) per eseguire queste trasformazioni.

Esempio: Il Cifrario di Cesare

The Caesar cipher 	A	B	C	D	E	F	G	H	I	L	M	N	O	P	Q	R	S	T	U	V	Z
	d	e	f	g	h	i	l	m	n	o	p	q	r	s	t	u	v	z	a	b	c

Il **cifrario di Cesare** è uno dei più antichi e semplici metodi di crittografia conosciuti. Prende il nome da Giulio Cesare, che si dice lo utilizzasse per comunicare con i suoi generali.

Come funziona:

Il cifrario di Cesare è un **cifrario a sostituzione monoalfabetica**. Questo significa che ogni lettera del testo in chiaro viene sostituita con un'altra lettera dell'alfabeto spostata di un certo numero di posizioni. Questo "spostamento" è la **chiave** del cifrario.

Esempio pratico:

Supponiamo di voler cifrare il messaggio: **CIAO**

E scegliamo una **chiave** di spostamento pari a **3**.

Per cifrare, spostiamo ogni lettera di 3 posizioni in avanti nell'alfabeto (considerando l'alfabeto in modo circolare, quindi dopo la 'Z' si ricomincia dalla 'A'):

- **C** spostata di 3 posizioni diventa **F** (C -> D -> E -> F)
- **I** spostata di 3 posizioni diventa **L** (I -> L -> M -> N)
- **A** spostata di 3 posizioni diventa **D** (A -> B -> C -> D)
- **O** spostata di 3 posizioni diventa **R** (O -> P -> Q -> R)

Quindi, il testo cifrato di "CIAO" con una chiave di 3 è: **FLDR**

Per decifrare:

Per decifrare il messaggio "FLDR" con la stessa chiave di 3, dobbiamo spostare ogni lettera di 3 posizioni **all'indietro** nell'alfabeto:

- **F** spostata indietro di 3 posizioni diventa **C** (F -> E -> D -> C)
- **L** spostata indietro di 3 posizioni diventa **I** (L -> K -> J -> I)
- **D** spostata indietro di 3 posizioni diventa **A** (D -> C -> B -> A)
- **R** spostata indietro di 3 posizioni diventa **O** (R -> Q -> P -> O)

Ritornando così al testo in chiaro originale: **CIAO**

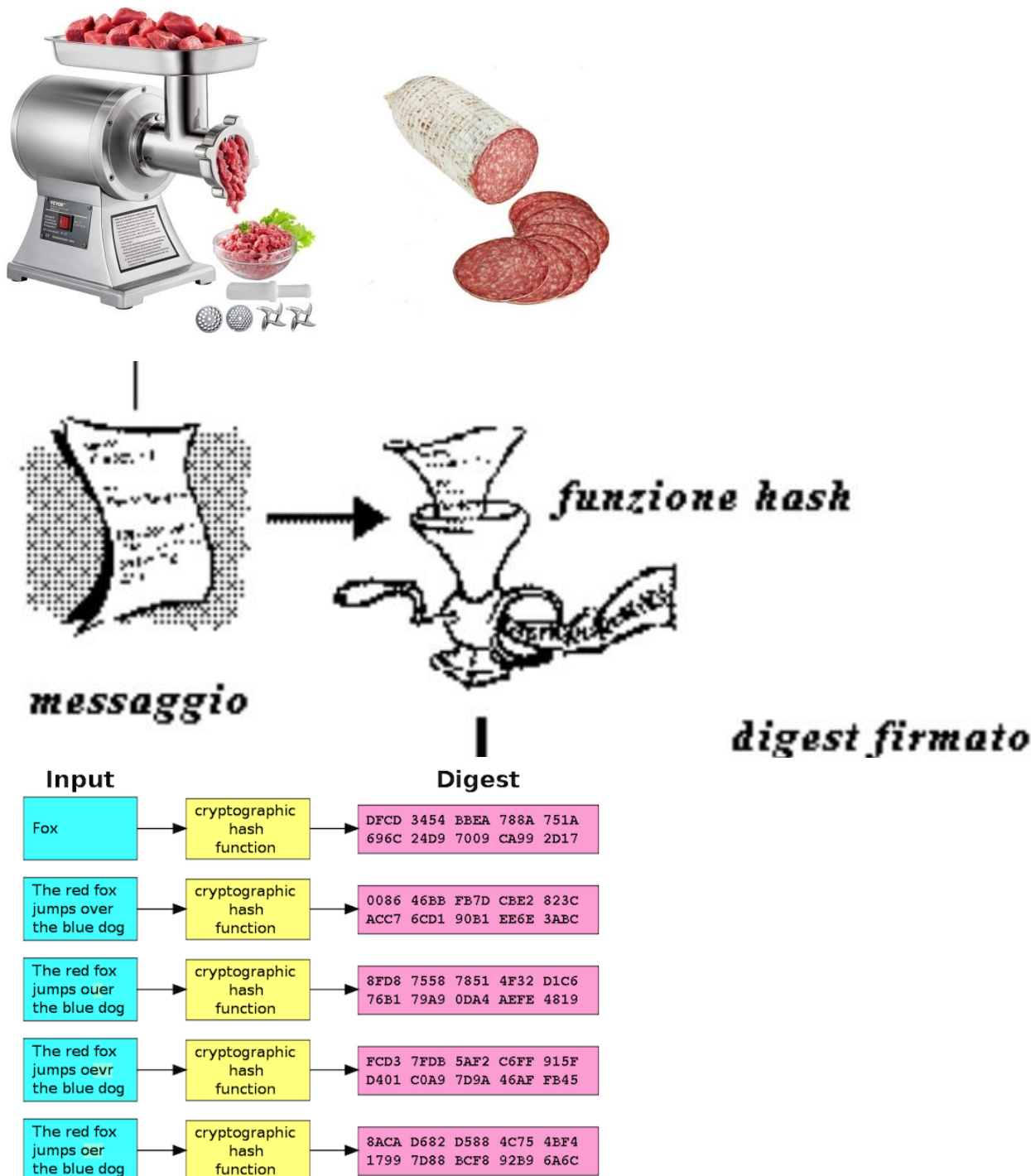
Limiti del Cifrario di Cesare:

Il cifrario di Cesare è molto semplice e facilmente decifrabile, soprattutto con tecniche di analisi delle frequenze delle lettere. Non è assolutamente sicuro per proteggere informazioni importanti al giorno d'oggi. Tuttavia, è un ottimo esempio per comprendere il concetto fondamentale di crittografia e il ruolo di una chiave.

Hashing

Analogia:

Immagina di avere un tritacarne molto potente (la funzione di hash SHA-256). Ci metti dentro qualsiasi cosa (il tuo input) e ne esce un salsicciotto di una lunghezza specifica (l'hash di 64 caratteri). È quasi impossibile, guardando solo il salsicciotto, capire esattamente cosa c'era dentro all'inizio. Inoltre, è estremamente improbabile che due insiemi completamente diversi di ingredienti producano esattamente lo stesso salsicciotto per forma, sapore e consistenza a livello molecolare.



Un **hash** è l'output di una funzione matematica chiamata **funzione di hash**. Questa funzione prende un input di dimensione arbitraria (che può essere un testo, un file, un insieme di dati, ecc.) e lo trasforma in un output di dimensione fissa, chiamato appunto "hash" o "valore hash".

Pensa a una funzione di hash come a un **compattatore di informazioni unidirezionale**. Prende qualcosa di potenzialmente grande e lo riduce a una stringa di caratteri di lunghezza predefinita. La caratteristica fondamentale è che è **molto difficile (idealmente impossibile)** risalire all'input originale partendo solo dall'hash.

Caratteristiche Importanti delle Funzioni di Hash:

- **Determinismo:** Dato lo stesso input, la funzione di hash produrrà sempre lo stesso output (hash).

- **Calcolo Veloce:** Deve essere computazionalmente efficiente calcolare l'hash di un dato input.
- **Resistenza alla Preimmagine (One-way):** Dato un hash, dovrebbe essere computazionalmente impraticabile trovare un input che produca quell'hash.
- **Resistenza alla Seconda Preimmagine:** Dato un input e il suo hash, dovrebbe essere computazionalmente impraticabile trovare un altro input diverso che produca lo stesso hash.
- **Resistenza alle Collisioni:** Dovrebbe essere computazionalmente impraticabile trovare due input diversi che producano lo stesso hash. Questa è la proprietà più forte e difficile da garantire perfettamente.

Hash SHA-256 Univoco:

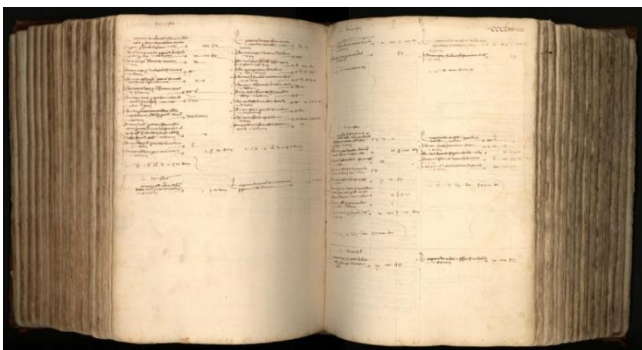
SHA-256 (Secure Hash Algorithm 256-bit) è una specifica **funzione di hash crittografica** appartenente alla famiglia SHA-2. Le funzioni di hash crittografiche sono progettate specificamente per avere le proprietà di resistenza menzionate sopra, rendendole adatte per applicazioni di sicurezza come la verifica dell'integrità dei dati, le firme digitali e la memorizzazione sicura di password.

Quando diciamo che un hash SHA-256 è **univoco** (o meglio, ha un'alta probabilità di essere univoco), intendiamo che:

- **L'output ha una dimensione fissa di 256 bit.** Questo si traduce in una stringa esadecimale di 64 caratteri. Indipendentemente dalla dimensione dell'input, l'output sarà sempre lungo 64 caratteri.
- **È estremamente improbabile trovare due input diversi che producano lo stesso hash SHA-256 (resistenza alle collisioni).** Sebbene teoricamente le collisioni esistano (perché ci sono infiniti input possibili e solo un numero finito di output di 256 bit), la probabilità di trovarne una casualmente è così bassa da essere considerata trascurabile per scopi pratici.

In sintesi, un hash SHA-256 è un'impronta digitale digitale di 256 bit di un dato. La sua "unicità" deriva dalla sua elevata resistenza alle collisioni, il che significa che è altamente improbabile che due dati diversi abbiano lo stesso hash SHA-256. Questa proprietà è fondamentale per la sicurezza e l'integrità dei sistemi che utilizzano la blockchain.

Blockchain come tipo di database



In un certo senso, una **blockchain può essere considerata un tipo di database**, ma con caratteristiche molto specifiche che la distinguono dai database tradizionali.

Ecco alcuni punti chiave per capire questa relazione:

Similitudini con un Database:

- **Memorizzazione di Dati:** Entrambi i sistemi sono progettati per memorizzare e organizzare informazioni (nel caso della blockchain, principalmente transazioni o dati).
- **Struttura Logica:** Entrambi hanno una struttura logica per organizzare i dati (tabelle nei database relazionali, blocchi concatenati nella blockchain).
- **Accesso ai Dati:** Entrambi permettono di accedere e recuperare i dati memorizzati, anche se le modalità di interrogazione possono essere molto diverse.

Differenze Fondamentali che Distinguono la Blockchain dai Database Tradizionali:

- **Decentralizzazione vs. Centralizzazione:**
 - I **database tradizionali** sono generalmente centralizzati, controllati da una singola entità o organizzazione che gestisce l'accesso e le modifiche.
 - La **blockchain** è intrinsecamente decentralizzata, distribuita su una rete di computer (nodi) dove ogni partecipante (o molti di essi) possiede una copia dell'intero registro. Non esiste un'autorità centrale unica.
- **Immutabilità vs. Modificabilità:**
 - Nei **database tradizionali**, i dati possono essere modificati, aggiornati ed eliminati da chi ha i permessi appropriati.

- Nella **blockchain**, una volta che un blocco di dati (una transazione) viene aggiunto alla catena, è estremamente difficile (praticamente impossibile senza un consenso massivo della rete) modificarlo o cancellarlo retroattivamente. Questo garantisce la sua **immutabilità**.
- **Trasparenza vs. Opacità (selettiva):**
 - La **trasparenza** è una caratteristica chiave di molte blockchain pubbliche (come quella di Bitcoin), dove tutte le transazioni sono visibili a tutti i partecipanti della rete, anche se l'identità degli utenti può essere pseudonima.
 - I **database tradizionali** possono avere vari livelli di accesso e trasparenza definiti dall'amministratore.
- **Sicurezza e Fiducia:**
 - La sicurezza dei **database tradizionali** dipende dalle misure implementate dall'entità che li controlla (firewall, controlli di accesso, crittografia, ecc.).
 - La **blockchain** raggiunge un elevato livello di sicurezza attraverso la crittografia (come l'hashing) e i meccanismi di consenso distribuiti. La fiducia non si basa su una singola autorità, ma sul consenso della rete.
- **Velocità e Scalabilità:**
 - I **database tradizionali** sono spesso ottimizzati per operazioni di lettura e scrittura veloci e possono gestire grandi volumi di dati in modo efficiente.
 - Le **blockchain**, a causa della necessità di raggiungere il consenso tra molti nodi e della natura sequenziale dell'aggiunta di blocchi, possono essere più lente e avere problemi di scalabilità (anche se sono in corso molte ricerche per migliorare questo aspetto).
- **Finalità:**
 - I **database tradizionali** sono progettati per un'ampia gamma di applicazioni di gestione dei dati.
 - La **blockchain** è particolarmente adatta per applicazioni che richiedono trasparenza, immutabilità, sicurezza e assenza di un'autorità centrale fidata (come le criptovalute, la tracciabilità della supply chain, i sistemi di voto sicuri, ecc.).

In conclusione:

Mentre la blockchain condivide con i database la funzione di memorizzare dati, le sue caratteristiche di decentralizzazione, immutabilità e il meccanismo di consenso la rendono un tipo di registro digitale fondamentalmente diverso dai database tradizionali. È più accurato considerarla un **registro distribuito e immutabile**, che può essere visto come una forma specializzata di database con proprietà uniche. Alcuni definiscono la blockchain come un "**database distribuito con garanzie di integrità e immutabilità**". Questa definizione cattura meglio la sua natura ibrida.

Piccola Blockchain in Python:

Sì, è assolutamente possibile creare una versione molto semplificata di una blockchain in Python per capire i concetti fondamentali. Ecco un esempio di base:

```
import hashlib
import datetime

class Block:
    def __init__(self, index, timestamp, transactions, previous_hash):
        self.index = index
        self.timestamp = timestamp
        self.transactions = transactions
        self.previous_hash = previous_hash
        self.hash = self.calculate_hash()

    def calculate_hash(self):
        block_string = str(self.index) + str(self.timestamp) +
str(self.transactions) + str(self.previous_hash)
        return hashlib.sha256(block_string.encode()).hexdigest()

class Blockchain:
    def __init__(self):
        self.chain = [self.create_genesis_block()]

    def create_genesis_block(self):
        return Block(0, datetime.datetime.now(), "Genesis Block", "0")

    def get_latest_block(self):
        return self.chain[-1]

    def add_block(self, new_block):
        new_block.previous_hash = self.get_latest_block().hash
        new_block.hash = new_block.calculate_hash()
        self.chain.append(new_block)

    def is_chain_valid(self):
        for i in range(1, len(self.chain)):
            current_block = self.chain[i]
            previous_block = self.chain[i-1]

            if current_block.hash != current_block.calculate_hash():
                return False

            if current_block.previous_hash != previous_block.hash:
                return False
        return True

# Esempio di utilizzo
my_blockchain = Blockchain()

# Aggiungiamo delle transazioni (semplificate)
my_blockchain.add_block(Block(1, datetime.datetime.now(), {"sender": "Alice",
"receiver": "Bob", "amount": 10}, ""))
my_blockchain.add_block(Block(2, datetime.datetime.now(), {"sender":
"Charlie", "receiver": "David", "amount": 5}, ""))
```

```
# Stampiamo la blockchain
for block in my_blockchain.chain:
    print("Index:", block.index)
    print("Timestamp:", block.timestamp)
    print("Transactions:", block.transactions)
    print("Previous Hash:", block.previous_hash)
    print("Hash:", block.hash)
    print("---")

# Verifichiamo se la catena è valida
print("Is chain valid?", my_blockchain.is_chain_valid())

# Tentiamo di manomettere un blocco (non funzionerà se la validazione è
corretta)
# my_blockchain.chain[1].transactions = {"sender": "Eve", "receiver":
"Mallory", "amount": 1000}
# print("Is chain valid after tampering?", my_blockchain.is_chain_valid())
```

Spiegazione del Codice:

1. **Block Class:** Rappresenta un singolo blocco nella blockchain. Contiene:
 - index: La posizione del blocco nella catena.
 - timestamp: Quando è stato creato il blocco.
 - transactions: Una lista di transazioni (in questo esempio, un semplice dizionario).
 - previous_hash: L'hash del blocco precedente nella catena.
 - hash: L'hash del blocco corrente, calcolato usando il metodo `calculate_hash`.
 2. **calculate_hash Method:** Genera un hash SHA-256 univoco per il blocco basandosi sul suo contenuto.
 3. **Blockchain Class:** Rappresenta l'intera blockchain. Contiene:
 - chain: Una lista di blocchi. Inizia con il "genesis block".
 4. **create_genesis_block Method:** Crea il primo blocco della catena (il blocco genesis).
 5. **get_latest_block Method:** Restituisce l'ultimo blocco aggiunto alla catena.
 6. **add_block Method:** Aggiunge un nuovo blocco alla catena. Prima di aggiungerlo, imposta il `previous_hash` del nuovo blocco all'hash dell'ultimo blocco e calcola l'hash del nuovo blocco.
 7. **is_chain_valid Method:** Verifica l'integrità della blockchain controllando che l'hash di ogni blocco sia corretto e che il `previous_hash` di ogni blocco corrisponda all'hash del blocco precedente.
- Questo è un esempio molto semplificato e non include funzionalità avanzate come il meccanismo di consenso (come Proof-of-Work o Proof-of-Stake) necessario in una vera blockchain per validare nuovi blocchi in un ambiente decentralizzato. Tuttavia, ti dà un'idea di come i blocchi sono concatenati e come l'hashing gioca un ruolo cruciale nella sicurezza.

Mining

"Mining" è un termine cruciale nel contesto delle blockchain, specialmente per quelle che utilizzano un meccanismo di consenso chiamato **Proof-of-Work (PoW)**, come Bitcoin.

In sostanza, il **"mining"** è il processo attraverso il quale nuovi blocchi di transazioni vengono aggiunti alla blockchain e, in cambio, i **"minatori"** (partecipanti al processo) vengono **ricompensati con nuove criptovalute e/o commissioni sulle transazioni incluse nel blocco**.

Ecco una spiegazione più dettagliata di cosa comporta il mining in una blockchain implementata, ad esempio, con Python (anche se le implementazioni reali su larga scala sono spesso in linguaggi più performanti come C++ o Go):

Il Processo di Mining:

1. **Raccolta delle Transazioni:** I minatori raccolgono le transazioni valide che sono state inviate alla rete e che non sono ancora state incluse in un blocco.
2. **Creazione di un Blocco Candidato:** Il minatore crea un nuovo blocco "candidato". Questo blocco contiene:
 - Un riferimento all'**hash** del blocco precedente nella catena, garantendo la sua integrità e la sequenza cronologica.
 - Un insieme delle transazioni raccolte.

- Un **nonce**: un numero arbitrario che i minatori modificheranno ripetutamente per risolvere il "puzzle" crittografico.
- Un **timestamp**.
- Un **target** (o "difficulty"): un valore che definisce quanto deve essere difficile trovare un hash valido per il blocco.

3. **Risoluzione del Puzzle Crittografico (Proof-of-Work):** La parte più intensa del mining è trovare un **hash** (un'impronta digitale unica) per l'intero blocco candidato che sia inferiore o uguale al **target**. Questo viene fatto provando un numero enorme di nonce diversi e calcolando l'hash del blocco ogni volta.

- **Hash:** Una funzione crittografica (come SHA-256, usata da Bitcoin) prende un input di qualsiasi dimensione e produce un output di dimensione fissa. È computazionalmente facile calcolare l'hash di un dato input, ma è computazionalmente molto difficile trovare un input che produca un hash specifico.
- **Nonce:** I minatori modificano il valore del nonce e ricalcolano l'hash del blocco. L'obiettivo è trovare un nonce che, quando incluso nel blocco e sottoposto alla funzione hash, produca un hash che inizi con un certo numero di zeri (il numero di zeri dipende dalla "difficulty").
- **Target/Difficulty:** La "difficulty" regola la velocità con cui vengono trovati nuovi blocchi. Se troppi blocchi vengono aggiunti troppo rapidamente, la difficulty aumenta, rendendo più difficile trovare un hash valido. Se i blocchi vengono aggiunti troppo lentamente, la difficulty diminuisce.

4. **Proof of Work Trovata:** Quando un minatore trova un nonce che produce un hash valido (inferiore al target), ha "risolto" il puzzle. Questo è la "prova di lavoro" (Proof-of-Work).

5. **Trasmissione del Blocco:** Il minatore che ha trovato la soluzione trasmette il blocco alla rete degli altri nodi.

6. **Validazione del Blocco:** Gli altri nodi verificano che:

- Il blocco sia formattato correttamente.
- Le transazioni al suo interno siano valide.
- L'hash del blocco sia effettivamente inferiore al target (verificando la proof-of-work).
- Il blocco si colleghi correttamente al blocco precedente tramite l'hash.

7. **Aggiunta alla Blockchain:** Se il blocco è valido, gli altri nodi lo aggiungono alla loro copia della blockchain. Questo rende il blocco parte della storia immutabile delle transazioni.

8. **Ricompensa:** Il minatore che ha creato il blocco viene ricompensato con una certa quantità di nuova criptovaluta (la "block reward") e con le commissioni pagate dagli utenti per le transazioni incluse nel blocco. Questa è la motivazione economica per i minatori di dedicare risorse al processo.

In Python:

Anche se il mining vero e proprio di blockchain consolidate come Bitcoin non viene fatto in Python a causa delle esigenze di performance, si possono creare **simulazioni** o **blockchain di prova concetto** in Python che implementano i concetti di base del mining. Questo può essere utile per scopi didattici o per sperimentare con diversi algoritmi di consenso.

In una simulazione Python, potresti avere funzioni per:

- Creare blocchi con transazioni, timestamp e un nonce.
- Implementare una funzione hash (dalla libreria hashlib).
- Implementare una funzione che, dato un blocco e un target, tenta diversi nonce fino a trovare un hash valido.
- Gestire la ricompensa per il minatore che trova la soluzione.

In sintesi, il mining è il processo computazionalmente intensivo di trovare una soluzione a un puzzle crittografico per aggiungere un nuovo blocco di transazioni alla blockchain, garantendo sicurezza e immutabilità, e ricompensando i partecipanti per il loro lavoro.

Termine Nonce, cosa significa?

Nel contesto del mining di blockchain (specialmente quelle basate su Proof-of-Work), il termine **"nonce"** si riferisce a un **numero arbitrario** che viene aggiunto all'intestazione di un blocco. Il suo scopo principale è quello di essere modificato ripetutamente dai minatori durante il processo di mining fino a quando non viene trovato un hash del blocco che soddisfa una determinata condizione di difficoltà (il "target").

Immagina il nonce come una sorta di "manopola" che i minatori possono girare per cambiare l'impronta digitale (l'hash) dell'intero blocco.

Ecco i punti chiave per capire il nonce:

- **"Number used only once":** L'acronimo "nonce" sta per "number used only once" (numero usato una sola volta). Anche se nel processo di mining vengono provati moltissimi nonce, ogni tentativo specifico di combinazione tra i dati del blocco e un particolare nonce produce un hash unico.

- **Parte dell'intestazione del blocco:** Il nonce è uno dei campi inclusi nell'intestazione di un blocco, insieme ad altre informazioni come l'hash del blocco precedente, il timestamp e la radice dell'albero di Merkle delle transazioni.
- **Variabile chiave nel Proof-of-Work:** Nel meccanismo di consenso Proof-of-Work, i minatori devono trovare un hash del blocco che sia inferiore o uguale a un valore target specificato dalla difficoltà della rete. Per fare ciò, modificano principalmente il valore del nonce e ricalcolano l'hash del blocco ad ogni tentativo.
- **Ricerca esaustiva (ma intelligente):** I minatori non hanno un modo diretto per "calcolare" il nonce corretto. Devono procedere per tentativi, provando un gran numero di nonce diversi (spesso in modo sequenziale o pseudo-casuale) e calcolando l'hash del blocco con ogni nuovo nonce.
- **La soluzione al "puzzle":** Quando un minatore trova un nonce che, combinato con gli altri dati del blocco, produce un hash che soddisfa il target di difficoltà, ha "risolto" il puzzle crittografico. Questo nonce diventa parte del blocco validato e dimostra che è stata eseguita una certa quantità di lavoro computazionale (la "proof-of-work").
- **Unicità per ogni blocco valido:** Ogni blocco valido nella blockchain avrà un nonce specifico che ha portato alla generazione dell'hash accettato dalla rete. È improbabile che un altro blocco valido abbia lo stesso identico nonce (anche se teoricamente possibile con una probabilità estremamente bassa).

In Python (in una simulazione di blockchain):

In una semplice implementazione di blockchain in Python a scopo didattico, potresti rappresentare un blocco come un dizionario o una classe. Il nonce sarebbe semplicemente una delle chiavi/attributi di questo blocco. La funzione di mining simulerebbe il processo di incrementare il nonce e calcolare l'hash del blocco fino a quando non viene trovato un hash che inizia con un certo numero di zeri (che rappresenta la difficoltà).

Ad esempio, in un codice Python semplificato:

```
import hashlib
import time

def calcola_hash(blocco):
    blocco_string = str(blocco).encode()
    return hashlib.sha256(blocco_string).hexdigest()

def mine_blocco(blocco, difficolta):
    nonce = 0
    while True:
        blocco['nonce'] = nonce
        hash_corrente = calcola_hash(blocco)
        if hash_corrente.startswith('0' * difficolta):
            return nonce, hash_corrente
        nonce += 1

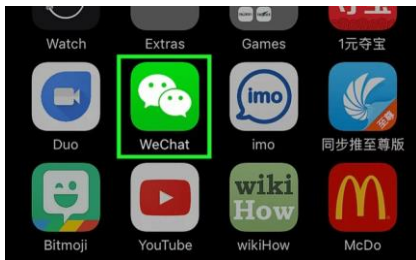
# Esempio di un blocco
blocco_precedente_hash = "0"
transazioni = ["Alice invia 1 BTC a Bob", "Charlie invia 0.5 BTC a David"]
timestamp = time.time()
blocco = {
    'indice': 1,
    'timestamp': timestamp,
    'transazioni': transazioni,
    'hash_precedente': blocco_precedente_hash,
    'nonce': 0 # Inizializzato a 0
}

difficolta = 2
nonce_trovato, hash_trovato = mine_blocco(blocco, difficolta)
print(f"Nonce trovato: {nonce_trovato}")
print(f"Hash del blocco: {hash_trovato}")
```

In questo esempio, la funzione `mine_blocco` incrementa il nonce del blocco fino a quando l'hash calcolato non inizia con due zeri (la difficoltà). Il nonce che porta a questo hash valido è la soluzione trovata dal "minatore".

In sintesi, il nonce è un elemento cruciale nel meccanismo di Proof-of-Work delle blockchain, agendo come un "contatore" che viene modificato per trovare un hash del blocco che soddisfi i requisiti di difficoltà della rete.

Wechat



piattaforma online con cui i cinesi fanno quasi tutto, dagli acquisti ai pagamenti alle ricerche, è **WeChat** (微信 - Wēixìn).

WeChat è molto più di una semplice app di messaggistica. È un vero e proprio **ecosistema digitale** o "super-app" che integra una vasta gamma di funzionalità, tra cui:

- **Messaggistica:** Chat testuali, vocali, videochiamate, condivisione di file e posizioni.
- **Social Network (Moments - 朋友圈):** Permette agli utenti di condividere aggiornamenti, foto e video con i propri contatti, simile a Facebook o Instagram.
- **Pagamenti (WeChat Pay - 微信支付):** Un sistema di pagamento mobile onnipresente in Cina, utilizzato per acquisti online e offline, trasferimenti di denaro tra utenti, pagamento di bollette e molto altro.
- **Mini Program (小程序 - Xiǎo Chéngxù):** "App all'interno dell'app" che consentono agli utenti di accedere a una miriade di servizi di terze parti senza dover scaricare e installare applicazioni separate. Questi includono e-commerce, prenotazioni di ristoranti e hotel, servizi di trasporto, giochi e molto altro.
- **Ricerca:** Anche se non è la sua funzione primaria, WeChat offre funzionalità di ricerca all'interno della piattaforma, consentendo agli utenti di trovare account ufficiali, contenuti e servizi.
- **Account Ufficiali (公众号 - Gōngzhòng Hào):** Aziende, media e organizzazioni possono creare account ufficiali per interagire con i propri follower, fornire informazioni, offrire servizi e vendere prodotti direttamente all'interno di WeChat.

La popolarità e la versatilità di WeChat lo rendono uno strumento indispensabile per la vita quotidiana in Cina. Gli utenti lo utilizzano costantemente per comunicare, socializzare, fare acquisti, pagare beni e servizi, accedere a informazioni e svolgere una miriade di altre attività, spesso senza mai uscire dall'applicazione.

Un'altra piattaforma di pagamento molto popolare in Cina è **Alipay (支付宝 - Zhīfùbǎo)**, di proprietà del gruppo Alibaba. Alipay è anch'essa ampiamente utilizzata per pagamenti online e offline e offre una vasta gamma di servizi finanziari. Tuttavia, WeChat integra anche forti funzionalità social e di comunicazione, rendendola la "super-app" per eccellenza con cui i cinesi gestiscono gran parte della loro vita digitale.