FSBM

FACULTE DES SCIENCES BEN M'SICK
UNIVERSITÉ HASSAN II DE CASABLANCA

INTRODUCTION À LA BLOCKCHAIN

DSBD-M1
2024/2025

Abdelaziz ETTAOUFIK
Enseignant Chercheur
FSBM
a.ettaoufik@gmail.com

PLAN

1. Introduction au Web 3.0
2. Introduction à la Blockchain
3. Éléments de cryptographie
4. Méthodes de consensus
5. La Blockchain en tant que structure de données

WEB 3

- Limites du Web 2
- Présentation du Web 3
- Avantages du Web 3
- Limites du Web 3
- Conclusion

WEB 3

Limites du Web 2

- La version d'Internet avec laquelle nous sommes tous familiers, générée par les utilisateurs et basée sur les réseaux sociaux
- Un Internet centralisé et dominé par les géants du Web tels que Google, Amazon, Apple et bien d'autres
- L'utilisation de ce système repose sur la confiance que doivent placer les utilisateurs en ces entreprises pour assurer l'intégrité de leurs données;
- Cependant, les fréquentes fuites de données, les hacks, le vol d'identité ou encore la vente de données sans consentement sont monnaie courante sur le Web 2.0

WEB 3

Limites du Web 2

- Les entreprises gérant les services utilisés ont un pouvoir de décision unilatéral, le contrôle est entre leurs mains.
- Le Web2 est ainsi dominé par ces entités centralisées, la liberté y est donc toute relative.
- Les géants du Web 2.0, notamment GAFA, collectent des quantités astronomiques de données sur leurs utilisateurs. Ces données sont ensuite monétisées par ces entreprises pour des campagnes publicitaires et autres pratiques afin de générer des profits.
- Il s'agit de l'un des problèmes fondamentaux du Web 2 puisque les utilisateurs ne sont pas récompensés pour partager ces données précieuses

PRÉSENTATION DU WEB 3

Le web 3.0 est la troisième génération des services Internet pour les sites et les applications. C'est l'Internet décentralisé, qui repose sur des technologies peer-to-peer comme la blockchain

- ✓ Il permet à chaque internaute de contrôler pleinement ses données personnelles

  Et

- ✓ De participer activement à la gouvernance du web.

La définition du web 3.0 a considérablement évolué depuis sa première formulation en 2008

« modèle qui permet aux données d'être partagées et réutilisées entre plusieurs applications, entreprises et groupes d'utilisateurs ». Tim Berners-Lee, Directeur W3C

PRÉSENTATION DU WEB 3

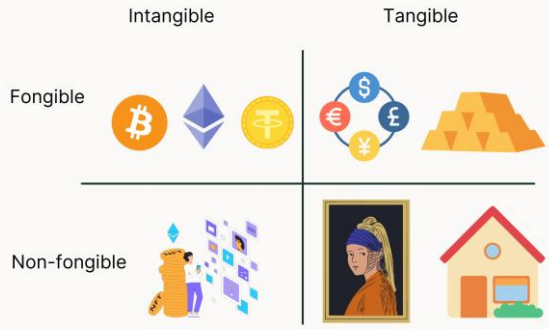
- Le Web 3 désigne les applications et plateformes décentralisées développées et opérant sur des blockchains publiques et parfois soutenues par des tokens non fongibles (NFTs).
- Ce sont des **dApps** (applications décentralisées), dont la technologie sous-jacente est la même que celle des cryptomonnaies.
- *Plusieurs définitions* ← le Web 3 est en train d'être construit

PRÉSENTATION DU WEB 3

On peut faire la comparaison avec des euros et une œuvre d'art pour mieux saisir ce concept :

- Un euro est égal à un euro. On peut les échanger sans problème, car ils ont la même valeur.
- En revanche, un tableau n'est pas égal à un autre tableau. On ne peut pas les échanger sans passer par une étape intermédiaire qui induit de la monnaie.

PRÉSENTATION DU WEB 3



Source : https://www.sortlist.fr/blog/wp-content/uploads/sites/3/2020/10/influenceurs-1.png

PRÉSENTATION DU WEB 3

Caractéristiques

l'élaboration du Web 3.0 devrait reposer sur des technologies comme :

- L'Intelligence artificielle
- Le Web sémantique
- La Blockchain

PRÉSENTATION DU WEB 3

Caractéristiques

- L'Intelligence artificielle
  - Permet de fournir des données plus pertinentes, et surtout plus rapidement, aux utilisateurs.
  - Ouvre la voie aux assistants virtuels.

PRÉSENTATION DU WEB 3

Caractéristiques

- Web sémantique
  - permet de catégoriser et de stocker des informations pouvant enseigner la signification de différents types de données.
  - Un site web peut alors interpréter des mots saisis par ses utilisateurs, de manière à créer et partager des contenus pertinents.

PRÉSENTATION DU WEB 3

Caractéristiques

- La Blockchain

Permet de créer et d'authentifier des contenus, de manière à les rendre uniques et certifiés

PRÉSENTATION DU WEB 3

Principes du Web 3

1. Décentralisation

- Contrairement à Web 2.0, la gestion du Web 3.0 est confiée à un réseau distribué d'utilisateurs comme les organisations autonomes décentralisées (DAO).
- Les participants de cette nouvelle forme de gouvernance doivent approuver collectivement les futures mises à jour, les transactions, les opérations, ...
- les dApps permettent de décentraliser l'Internet et le rendre plus juste, transparent, fiable et résistant à la censure.

  → Aucune entité centrale n'a le pouvoir de prendre une décision unilatéralement et c'est l'une des plus grandes forces du Web3.

PRÉSENTATION DU WEB 3

Principes du Web 3

1. Décentralisation (suite)

- Ce principe de décentralisation peut s'appliquer sur un grand nombre de sujets avec la logique des Smart Contracts.

PRÉSENTATION DU WEB 3

Avantages du Web 3

2. Contrôle

Chaque copie d'un fichier doit être identique, et si elle ne l'est pas, les données contenues dans ce fichier deviennent invalides.

PRÉSENTATION DU WEB 3

Avantages du Web 3

3. Confidentialité

Ni entité ni individu n'a la possibilité d'accéder ou de changer les données d'un fichier sans la permission de la personne qui en est le propriétaire, ou l'accord de la majorité du réseau distribué.

PRÉSENTATION DU WEB 3

Avantages du Web 3

4. Disponibilité

Grâce au réseau distribué, même si un serveur tombe en panne ou cesse son activité les données seront toujours accessibles via d'autres serveurs sur lesquels elles sont stockées.

PRÉSENTATION DU WEB 3

Web 3 vs Web 2

	Web 2.0	Web 3.0
Serveurs	Centralisés	Décentralisés
Sécurité	Garantit par les entreprises	Garantit par la blockchain
Les données des utilisateurs	Les entreprises en ont le contrôle	L'utilisateur en a le contrôle
Censure	Possible, unilatéralement	Résistant à la censure
Création de contenu	Dépendance au média	Aucune dépendance
Propriété de l'identité et du contenu créé	Entreprises	Utilisateurs
Accès aux services	Avec permission (des entreprises)	Sans permission

LA TECHNOLOGIE BLOCKCHAIN

INTRODUCTION

1. Présentation de la blockchain
2. Terminologie Blockchain
3. Distinction entre les bases de données et les registres de blockchain

INTRODUCTION

Blockchain

- Aussi nommées "DLT" (Distributed Ledger Technology – un registre de compte partagé rudimentaire)
- Un registre qui contient les transactions (d'une monnaie virtuelle par exemple).
- La blockchain grandit lorsque l'on rajoute des blocs.
- Les transactions ajoutées sont traitées par des ordinateurs connectés au réseau (nœud)
- Les blocs sont ajoutés au réseau en ordre chronologique
- La blockchain Ethereum permet d'y héberger des smart contrats avec l'Ether comme monnaie.
- La blockchain est désignée de façon à être durable et n'est pas contrôlée par une entité.

INTRODUCTION

Blockchain

Techniquement :

- Une base de données distribuée – un registre publique

  → il est possible d'insérer ou de rechercher des données, mais pas de les mettre à jour ou les supprimer)
- Un ordinateur distribué – qui exécute des contrats intelligents
- Basé sur les technologies P2P (pair-à-pair), la cryptographie et des API

INTRODUCTION

Pourquoi la blockchain ?

- Un fichier peut être dupliqué lorsqu'il est transmis. Cependant, lorsque la quantité transférée est un actif numérique, il est important de s'assurer que celui qui transmet l'actif n'en garde pas une copie.

  → problème de la double dépense.

  Ce problème a été résolu pour la première fois par la blockchain Bitcoin.
- Supprimer les intermédiaires
  - Notaires
  - Avocats
  - Comptables

INTRODUCTION

Révolution technologique

- Transmission de valeur (propriétés, voitures et autres biens) de façon sécurisée grâce aux contrats intelligents.
- Plus d'intermédiaires dans le processus de vente
- Transfert de l'argent rapide dans le monde entier
- Plus de capacité grâce à ces milliers de nœuds qui travaillent dans le même réseau.

INTRODUCTION

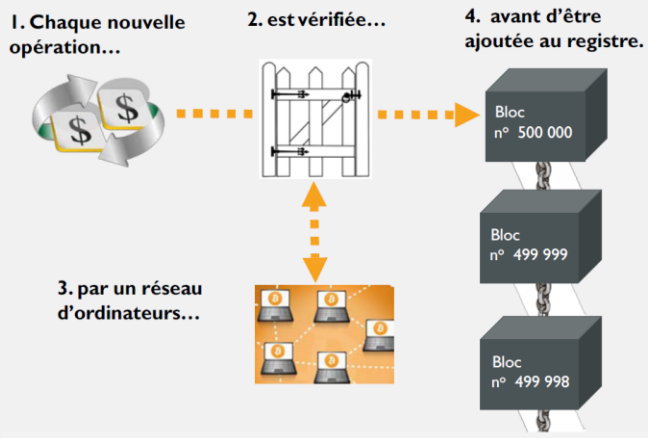
Blockchain

En fait, les blockchains forment plus qu'une technologie, elles :

- Contiennent habituellement des transactions financières
- Sont répliquées à travers un grand nombre de systèmes en quasi-temps réel
- Utilisent la cryptographie et les signatures numériques pour prouver l'identité des acteurs, l'authenticité des transactions, et faire respecter les droits d'accès en lecture/écriture
- Peuvent être accédées en écriture par un certains nombre de participants
- Peuvent être lues par les participants, habituellement un cercle plus large que pour les droits en écriture
- Possèdent des mécanismes pour rendre difficile le changement des données historiques, ou du moins rendent facile la détection d'une tentative de le faire

INTRODUCTION

Blockchain



ÉVOLUTION TECHNIQUE DE LA BLOCKCHAIN

	Blockchain 1.0	▪ Cryptocurrency ▪ Distributed ledger ▪ Consensus algorithm
	Blockchain 2.0	▪ Smart contracts ▪ Distributed & decentralised applications
HYPERLEDGER r3.c·rda	**Blockchain 3.0**	▪ Dapps ▪ Enterprise blockchain ▪ Scalability ▪ Inter-operability ▪ Efficiency
	Blockchain 4.0	▪ Industry infrastructure based Blockchain Ecosystem

Terminologie

- Nœud
- Ledger
- Genesis Block
- Transaction
- Mineur- Minage
- REWARD
- Fork
- Nonce
- Solidity
- Difficulty
- Consensus

TERMINOLOGIE

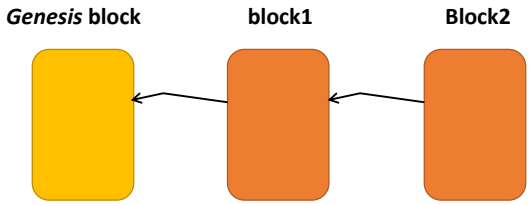
Nœud

- Une blockchain est déployée sur un ensemble d'ordinateurs. Chaque ordinateur constitue un nœud.
- On distingue deux types de nœuds :
  - ✓ Nœuds complets : contiennent une copie complète de la blockchain.
  - ✓ Nœuds légers : qui hébergent une version minimale de la blockchain leur permettant de participer aux échanges sur le réseau.

TERMINOLOGIE

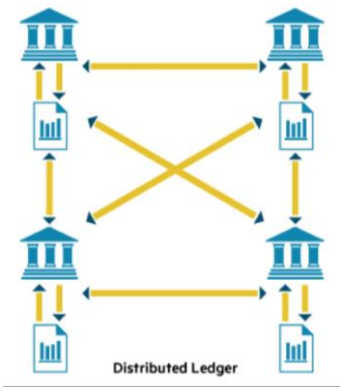
Ledger

- Peut être décrit comme la base de données qui enregistre toutes les transactions dans la blockchain depuis le jour 0. Il est implémenté comme une chaîne de blocs reliés entre eux au bloc de genèse (genesis block).
- Peut être :
  - ✓ Centralisé / Décentralisé
  - ✓ Publique / Privé



TERMINOLOGIE

Ledger distribué



TERMINOLOGIE

Ledger distribué

Les utilisateurs initient les transactions en les signants numériquement

Les utilisateurs diffusent leurs transactions aux noeuds du réseau

Un ou plusieurs noeuds commencent à valider chaque transaction

Les noeuds aggrègent les transactions validées dans des *blocs*

Les noeuds diffusent les blocs entre eux

Le protocol de consensus est utilisé

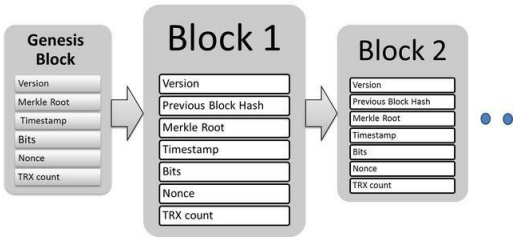
Le bloc représentant "l'état réel" est chaîné au *bloc* précédent

INTRODUCTION

Genesis Block

Représente le premier block du blockchain.

- Les versions modernes du Bitcoin utilise le block 0
- Les versions très anciennes le comptaient comme bloc 1

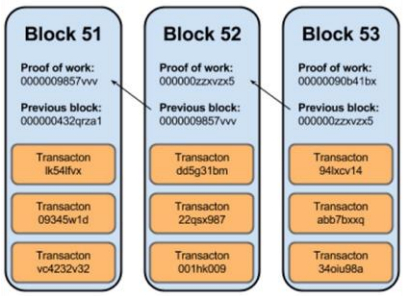


Un exemple d'un genesis block suivi de deux blocks (Block 1 et Block 2).

TERMINOLOGIE

Transactions et blocs

- Un bloc de transaction est une collection de transactions sur le réseau.
- Le bloc (groupe de transactions) est ajouté à la blockchain. Il peut être hashé.



INTRODUCTION

Transactions

- Le registre d'une blockchain est constitué de transactions.
- La transaction la plus simple est l'échange d'actifs numériques d'un nœud à l'autre.
- Une transaction peut aussi se faire d'un nœud vers un contrat intelligent dans le cadre de l'exécution d'un contrat intelligent sur une blockchain programmable comme Ethereum.

INTRODUCTION

Transactions

Une transaction se fait entre deux adresses avec un certain montant.

Données d'une transaction :

- From :[compte]
- To :[compte]
- Valeur : [integer] en Wei (1 Ether= 10^{18} Wei)
- Gas : le montant max de gaz utilisé
- gasPrice : le montant de Wei par gaz
- Data : ABI string byte
- Nonce : [integer] qui incrémenté à chaque transaction de type replay

TERMINOLOGIE

Mineurs

- Sont les ordinateurs qui opèrent le réseau.
- Hébergent une copie de la blockchain
- Ajoutent de nouveaux blocs à la Blockchain
- Vérifient l'intégrité de la blockchain
- Exécutent les contrats intelligents.

TERMINOLOGIE

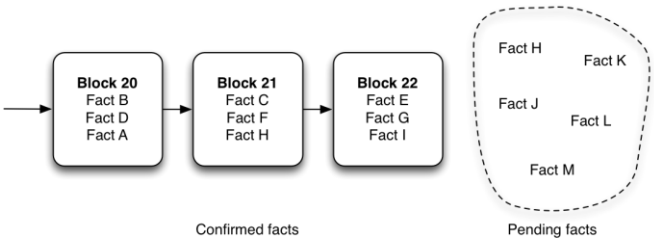
Mineurs

- Les blocs, reproduits sur plusieurs ordinateurs, représentent collectivement l'historique (c'est-à-dire les registres distribués de toutes les transactions survenues sur le réseau Bitcoin et reliées de façon cryptographique).
- Des incitatifs sont en place pour que ces ordinateurs *rivalisent* les uns contre les autres et, en même temps, coo*pèrent* les uns avec les autres!
- Ils coopèrent pour maintenir le réseau, mais chaque ordinateur rivalise avec les autres pour être le premier à *miner* un nouveau bloc (c'est-à-dire résoudre un problème complexe qui permet de former un nouveau bloc et d'être récompensé pour cela).
- Cette tenue de registres assure la sécurité et l'intégrité du système.

TERMINOLOGIE

Minage

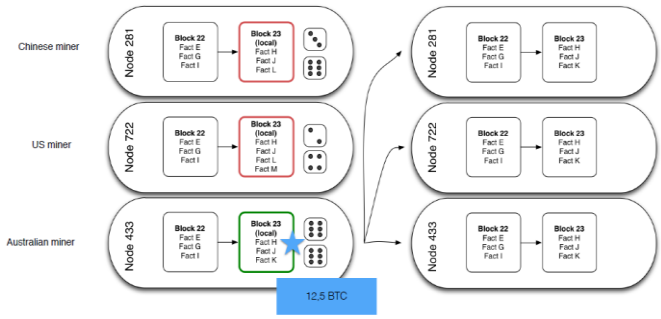
- Le minage d'une crypto est la validation, par les mineurs, de la transaction réalisée en monnaie numérique.
- Le processus par lequel les transactions sont vérifiées et ajoutées à la blockchain



TERMINOLOGIE

Minage

Ce processus, correspondant à la résolution de problèmes cryptographiques par l'utilisation de puissance de calcul, déclenche également la création de nouvelles cryptomonnaies.



TERMINOLOGIE

Les forks, ou *bifurcations*

- Ce mot anglais est utilisé pour désigner un embranchement, cad, une division en plusieurs branches.
- Au niveau de la blockchain, il désigne une division de la chaîne de blocs en deux chaînes distinctes.
- Un fork peut être provoqué par la divergence des règles de consensus, auquel cas on parle de **hard fork** (« embranchement dur ») qui désigne une séparation de la chaîne qui est généralement permanente
- hard fork est aussi utilisé pour désigner un changement non rétrocompatible des règles de consensus qui peut provoquer ce type d'embranchement

TERMINOLOGIE

Les forks, ou *bifurcations*

- La création d'une version alternative de la blockchain, en créant deux blocs simultanément à deux endroits différents du réseau. Cela créé deux blockchain parallèles, parmi lesquelles une sera gagnante sur l'autre.
- Ce qui signifie que les anciens nœuds n'accepteront généralement pas les blocs créés par les nouveaux nœuds.
- Quand-est ce que cela se produit t'il ?
  - Si deux blocs sont trouvés au même moment par deux mineurs différents
  - En cas d'incompatibilités entre les logiciels des nœuds des mineurs
  - En cas de désaccord sur les règles du protocole entre plusieurs nœuds

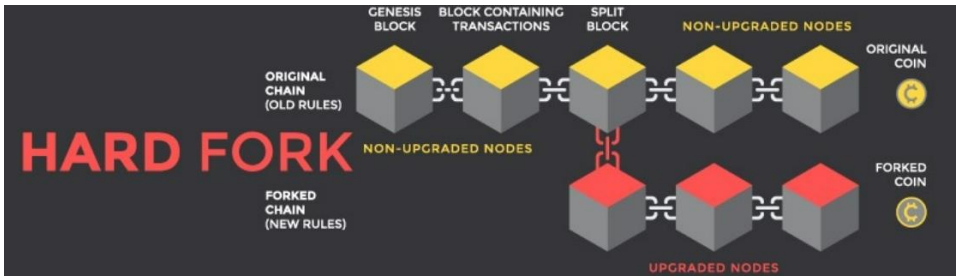
TERMINOLOGIE

Les forks, ou *bifurcations*

- Un hard fork (ou hardfork) est un changement radical du protocole d'un réseau qui rend valides des blocs et des transactions auparavant invalides, ou vice-versa.
- Un hard fork nécessite que tous les nœuds ou utilisateurs mettent à niveau vers la dernière version du logiciel de protocole.
- Les forks peuvent être initiés par des développeurs ou des membres d'une communauté cryptographique qui ne sont plus satisfaits des fonctionnalités offertes par les implémentations de blockchain existantes.

TERMINOLOGIE

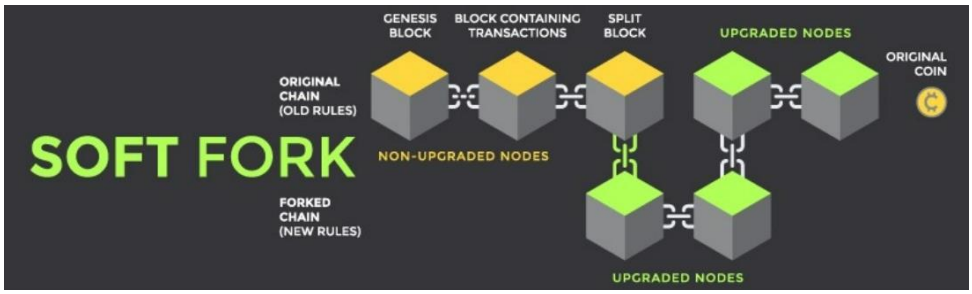
Forks



Source : https://www.financemagnates.com/cryptocurrency/education-centre

TERMINOLOGIE

Forks

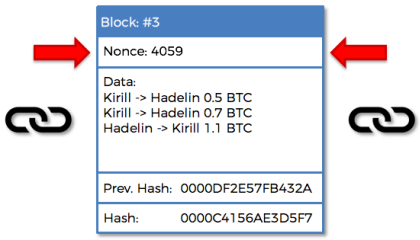


Source : https://www.financemagnates.com/cryptocurrency/education-centre

TERMINOLOGIE

Nonce (number only used once)

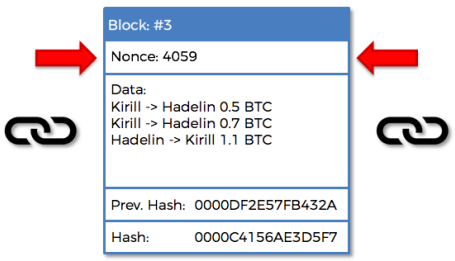
- le nonce est un petit morceau de données dans le bloc qui peut être modifié de manière aléatoire et répétée tout le temps afin que les mineurs puissent continuer à hacher les données de l'ensemble du bloc
- les mineurs continuent de deviner des nonces aléatoires jusqu'à ce qu'ils trouvent un nonce qui, avec le reste des données du bloc, générera la sortie (signature) qui répond aux termes du bloc



TERMINOLOGIE

Nonce (number only used once)

- Les nonces sont générés souvent pour modifier le résultat d'une fonction dans une communication cryptographique
- Il s'agit d'un horodatage ou d'un marqueur spécial destiné à empêcher les reproductions non autorisées d'un fichier



TERMINOLOGIE

Solidity

- Est un langage orienté objet pour la création des smart contrats
- Il a été développé pour permettre l'écriture de contrats intelligents sur des plates-formes blockchain telles qu'Ethereum

TERMINOLOGIE

REWARD

- Un block reward se traduit en français par « récompense de block ». Dans les cryptomonnaies, c'est une récompense qui est attribuée aux mineurs qui ont ajouté un bloc sur la blockchain.
- Selon les actifs numériques considérés, l'attribution de ces récompenses peuvent varier. Cependant, dans la plupart des cas, les personnes qui insèrent un bloc dans la blockchain reçoivent des tokens de la crypto-monnaie pour laquelle ils ont décidé de devenir mineur.
- Actuellement, Reward vaut 3,125BTC (Mai 2025)

TERMINOLOGIE

Difficulté

- La difficulté est une mesure de la difficulté à exploiter un bloc Bitcoin, ou en termes plus techniques, à trouver un hachage en dessous d'une cible donnée.
- Une difficulté élevée signifie qu'il faudra plus de puissance de calcul pour exploiter le même nombre de blocs, ce qui rend le réseau plus sûr contre les attaques.
- La difficulté est ajustée tous les 2016 blocs (toutes les 2 semaines environ) afin que le temps moyen entre chaque bloc reste de 10 minutes.
- La difficulté vient directement des blocs de données confirmés dans le réseau Bitcoin.

TERMINOLOGIE

Consensus

- Un procédé par lequel les nœuds d'un réseau pair à pair se mettent d'accord sur un ensemble d'informations.
- Dans le contexte des cryptomonnaies, un tel algorithme permet aux nœuds d'être en accord sur le registre des transactions.
- Le plus célèbre d'entre eux, qui est à la base de Bitcoin, est le consensus de Nakamoto par preuve de travail (proof of work en anglais).

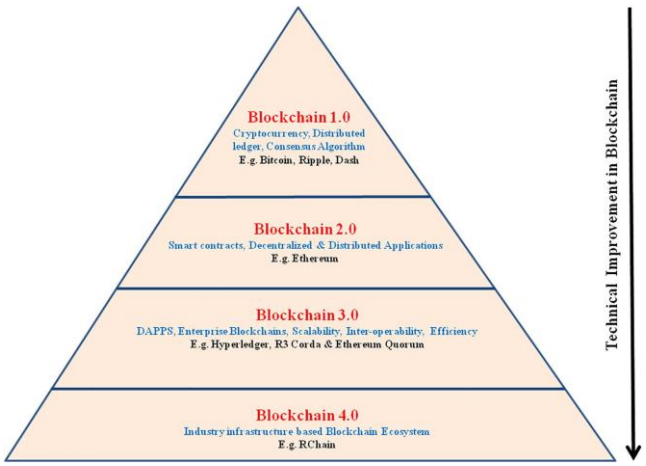
  → Celui-ci se base sur le principe suivant : chaque bloc est ajouté à la chaîne grâce à une dépense énergétique, c'est la preuve de travail.

COMPOSANTS DE LA BLOCKCHAIN

La Blockchain présente les cinq éléments suivants :

- ✓ Cryptographie;
- ✓ Réseau poste à poste: ordinateurs distribués, connectés les uns aux autres et communiquant entre eux;
- ✓ Mécanisme de consensus: façon dont tous ces ordinateurs peuvent se mettre d'accord;
- ✓ Registre: liste des transactions;
- ✓ Règles de validité: règles dictant quelles transactions sont valides et lesquelles ne le sont pas.

EVOLUTION DE LA BLOCKCHAIN



https://www.sciencedirect.com/science/article/pii/S131915782100207X

IMPLÉMENTATION DE BLOCKCHAIN

Publique

- Bitcoin. Monnaie numérique décentralisée
- Ethereum. Smart contracts et dApps
- Litecoin. Paiements plus rapides que Bitcoin
- Cardano. Plateforme évolutive de smart contracts
- Polkadot. Interopérabilité entre blockchains

Privé

- Hyperledger Fabric
- R3 Corda
- Quorum

Hybride

- Dragonchain
- Multichain

IMPLÉMENTATION DE BLOCKCHAIN

Bitcoin

- Un registre distribué
- La cryptomonnaie Bitcoin est le premier actif basé sur la technologie Blockchain
- Utilisée pour la vente en ligne de drogues et armes illégaux, ainsi que les *ransomware* (rançongiciel)
- Utilisée pour réaliser des transferts de fonds, de la spéculation, et en tant que réserve de valeur

"Ce dont on a besoin est un système de paiement électronique basé sur des preuves cryptographiques et non sur la confiance, qui permettrait à deux parties de réaliser des transactions directement entre elles, sans avoir besoin de tierce partie de confiance."

Satoshi Nakamoto – 31 Oct. 2008

IMPLÉMENTATION DE BLOCKCHAIN

Bitcoin

- À l'intérieur de la blockchain Bitcoin
  - **Entête du bloc** : contient des données techniques, le hash du bloc précédent, racine de Merkle, horodatage, difficulté, nonce

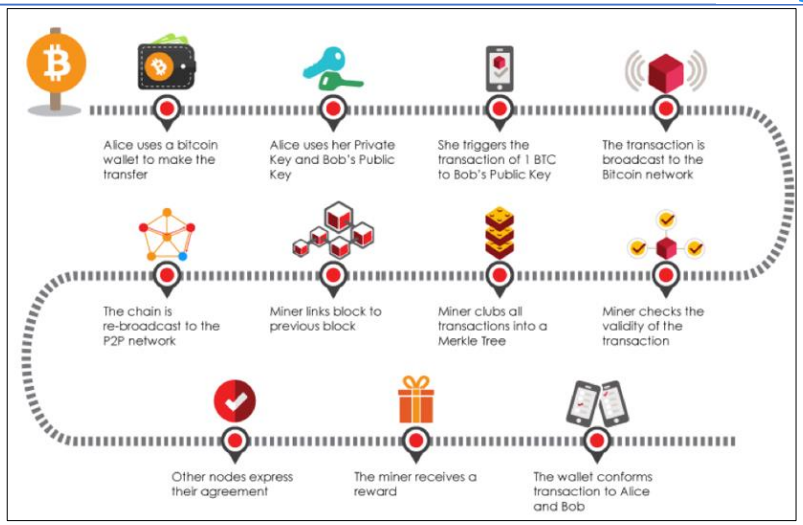
Exemple :

Height	448909
Block time	2017-01-19 09:32:58
Trades sum	5,340.87080329 BTC
Nb txs	1637
Difficulty	336,899,932,795.81
Fee	0.41239309 BTC
Hash	000000000000000000dbc285314939baad1f09d086fa68a0105d79378bf7629
Version	127
Confirmations	1
Merkle root	a4772eff88cbe645bba832d31730f0b42ea4d8d05d02ea62be533316bd3fb197
Prev block hash	000000000000000001527f8f089845eaa41753e61a0f97c54b364325ca74a6275
Size	947.32 kB
Coin days destroyed	2,913.95

IMPLÉMENTATION DE BLOCKCHAIN

Bitcoin

- Comment le transfert de monnaie fonctionne?



IMPLÉMENTATION DE BLOCKCHAIN

Ethereum

- Une plateforme informatique distribuée
- Proposé fin 2013 par Vitalik Buterin (chercheur et programmeur dans le domaine des cryptomonnaies)
- Financement participative par vente d'Ether contre des bitcoin, durant l'été 2014

"Une blockchain est un ordinateur magique sur lequel tout le monde peut télécharger des programmes, et les faire s'auto-executes, où l'état présent ainsi que l'historique des états de tous les programmes sont toujours visibles publiquement, et qui intègre une garantie crypto-économique très forte que ces programmes continueront de s'exécuter exactement de la manière spécifiée par le protocole de la blockchain." Vitalik Buterin

IMPLÉMENTATION DE BLOCKCHAIN

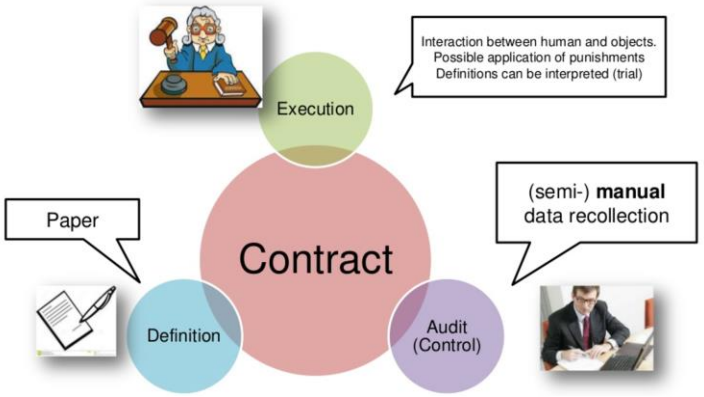
Ethereum

- Plateforme d'applications décentralisées (appelées Dapps)
- Registre de transactions et de contrats intelligents
- Basé sur l'Ethereum Virtual Machine (EVM), machine virtuelle d'Ethereum
- Intègre une cryptomonnaie, appelée Ether (ETH)

IMPLÉMENTATION DE BLOCKCHAIN

Ethereum

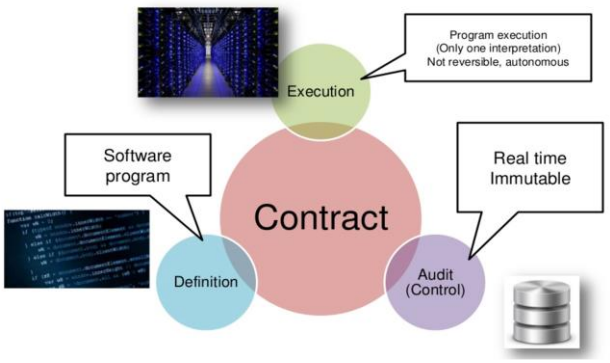
Comment fonctionne un contrat "traditionnel"?



IMPLÉMENTATION DE BLOCKCHAIN

Ethereum

Comment fonctionne un contrat "intelligent" ?



IMPLÉMENTATION DE BLOCKCHAIN

Ethereum

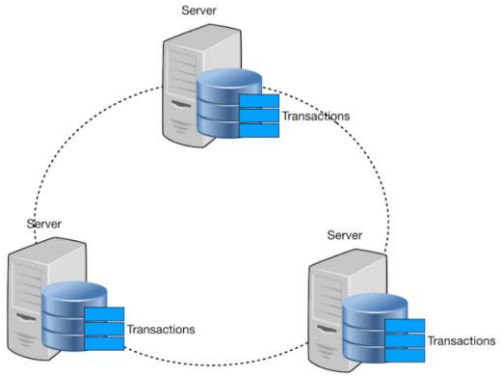
- Chaque compte dans le réseau Ethereum possède une clé privée et une clé publique
- Chaque compte Ethereum est représenté par une adresse qui est simplement un hash de la clé publique
- Les comptes peuvent utiliser leur clé privée pour signer une transaction
- Tout le monde peut vérifier la signature générée pour récupérer la clé publique/l'adresse du signature, et vérifier l'intégrité du message signé par la signature
- La clé publique peut être récupérée à partir de la clé privée

BLOCKCHAIN

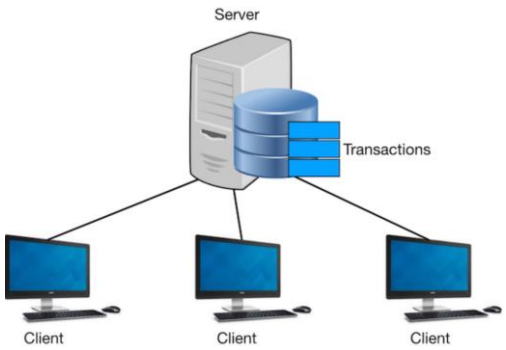
DISTINCTION ENTRE LES BASES DE DONNÉES ET LES REGISTRES DE BLOCKCHAIN

DISTINCTION ENTRE LES BASES DE DONNÉES ET LES REGISTRES DE BLOCKCHAIN

- Au niveau de l'architecture



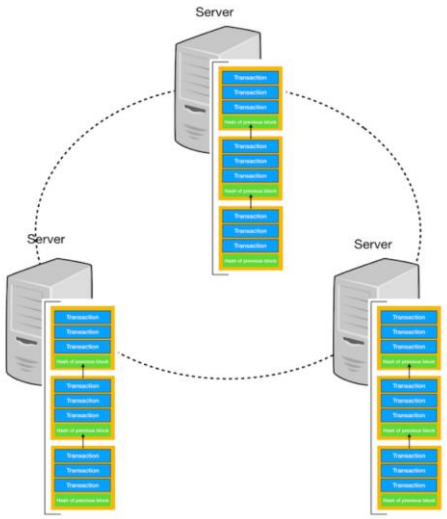
Registres de blockchain



Bases de données

DISTINCTION ENTRE LES BASES DE DONNÉES ET LES REGISTRES DE BLOCKCHAIN

- Au niveau de l'architecture



Registres de blockchain

Distinction entre les bases de données et les registres de blockchain

Bases de données	VS	Blockchains
Les bases de données ont des administrateurs, et un contrôle centralisé		Personne n'a le contrôle ou administre une blockchain
Uniquement les individus ayant les droits ont accès à la base de données		Tout le monde peut accéder à une blockchain (publique)
Seulement les individus autorisés peuvent lire et/ou écrire sur la base de données		N'importe qui, qui possède une prévue de travail valide, peut écrire sur la blockchain
Les bases de données sont rapides		Les blockchains sont lentes
Il n'y a pas de registre de transactions contenant tout l'historique		Présence d'un registre de transactions avec tout l'historique

ÉLÉMENTS DE CRYPTOGRAPHIE

- Cryptographie,
- Fonctions de hashage
- Signatures numériques

ÉLÉMENTS DE CRYPTOGRAPHIE

Cryptographie

Le chiffrement et déchiffrement des données

2 concepts principaux de cryptographie utilisée dans la technologie Blockchain :

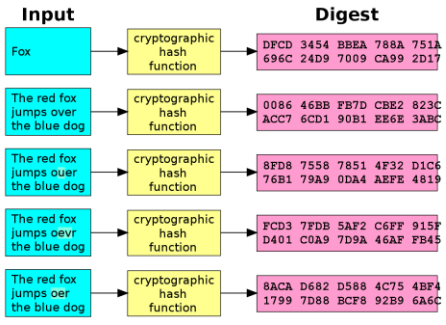
- Les fonctions de hachage
- Les signatures numériques

Il existe 3 formes de chiffrement largement utilisées :

Cryptographie symétrique	Cryptographie asymétrique	Hachage
Le même mot de passe sert à chiffrer et déchiffrer les données	Un mot de passe sert à chiffrer les données, un autre à les déchiffrer	Projette vers un espace de dimension fixe
Fonction bidirectionnelle	Les mots de passe vont par pair (clé publique / clé privée)	Fonction monodirectionnelle

HACHAGE CRYPTOGRAPHIQUE

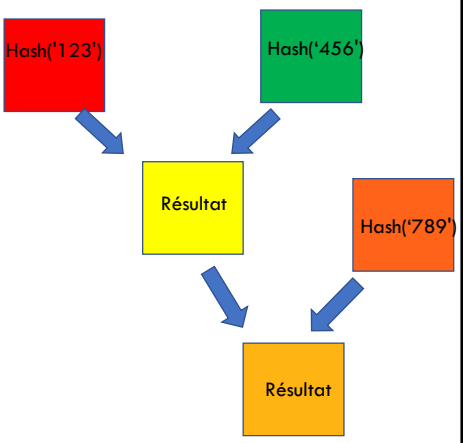
- La même donnée donnera la même sortie, on dit que c'est déterministe
- La sortie est rapide à calculer
- Il n'est pas possible de trouver l'entrée à partir de la sortie
- Le moindre changement de l'entrée va complétement changer la sortie



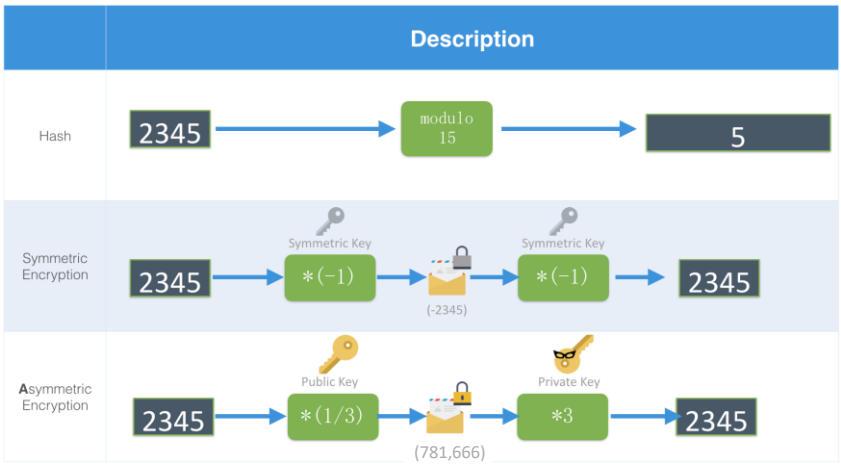
Source : https://fr.wikipedia.org/wiki/Fonction_de_hachage_cryptographique

HACHAGE CRYPTOGRAPHIQUE

- Le bloc jaune provient du bloc rouge et vert
- Le block orangé provient du bloc jaune et orange
- Si on change la couleur du bloc rouge tous les blocks suivant devraient changer de couleur
- Tous les blocs sont liés par leur hach, et sont copiés sur les nœuds du réseau.
- Si on veut falsifier une donnée, il faudrait changer tous les blocs sur tous les nœuds du réseau.



HACHAGE CRYPTOGRAPHIQUE



DÉMO

https://andersbrownworth.com/blockchain/hash

MÉTHODES DE CONSENSUS

1. Principes et paradigmes des systèmes distribués
2. Algorithmes de consensus Blockchain

CONSENSUS

Dans un réseau d'entreprises dont les participants sont connus et dignes de confiance, les transactions peuvent être vérifiées et inscrites dans le registre par différents moyens de *consensus* (accord), en particulier de la manière suivante :

- **Preuve de participation :** Pour valider les transactions, les validateurs doivent détenir un certain pourcentage de la valeur totale du réseau.
- **Signatures multiples :** Une majorité de validateurs (par exemple trois sur cinq) doivent s'accorder sur la validité d'une transaction.
- **Règlement des litiges (tolérance aux divergences - PBFT) :** Un algorithme permet de régler les litiges entre *nœuds* informatiques lorsqu'un nœud d'un ensemble produit des informations différentes des autres.

MÉTHODES DE CONSENSUS

1. Principes et paradigmes des systèmes distribués
   - ***Tolérance aux pannes byzantines*** (Byzantine Fault Tolerance - BFT) : la fiabilité d'un système tolérant aux pannes, en particulier les systèmes distribués, où des composants peuvent faire défaut, et l'information de leur panne est imparfaite à travers le réseau.
   - L'objectif de la BFT est de se défendre contre la panne de composants du système, avec ou sans symptômes qui empêcheraient les autres composants d'arriver à un consensus entre eux, lorsque ce consensus est nécessaire au bon fonctionnement du système.
   - Un exemple de BFT utilisé est bitcoin. Le réseau bitcoin fonctionne en parallèle pour générer une blockchain avec preuve de travail proof-of-work permettant au système de surmonter les pannes Byzantines et d'arriver à une vue cohérente globale de l'état du système.

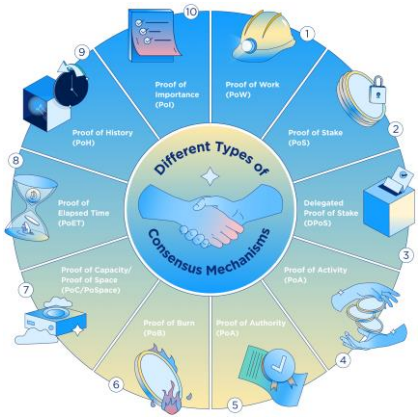
MÉTHODES DE CONSENSUS

2. Algorithmes de consensus Blockchain

- Derrière chaque cryptomonnaie, il se cache un algorithme de consensus. Aucun de ces algorithmes n'est parfait, mais chacun à ses forces.
- Dans le monde de la crypto, ces algorithme sont utilisés pour résoudre le problème de double dépense : comment s'assurer qu'un utilisateur n'utilise pas deux fois la même cryptomonnaie pour payer deux personnes différentes ?
- Exemple d'algorithmes :
  - Preuve de travail - Proof of Work (PoW)
  - Preuve d'enjeu - Proof of Stake (PoS)
  - Preuve d'enjeu déléguée - Delegated Proof of Stake (DPOS)
  - Preuve de brulage - Proof of Burn (PoB)
  - Practical Byzantine fault tolerant Mechanism (PBFT)
  - …

TERMINOLOGIE

Algorithmes de consensus Blockchain

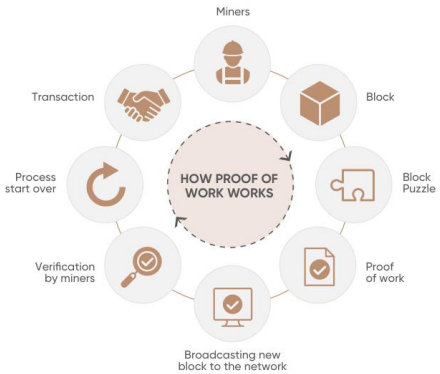


Source : THE IMPACT OF BLOCKCHAIN TECHNOLOGY ON IMPROVING CYBERSECURITY MEASURES, *International Research Journal of Modernization in Engineering Technology and Science.* Volume:06/Issue:06/June-2024

PROOF OF WORK

- les mineurs s'affrontent essentiellement pour résoudre des énigmes informatiques extrêmement complexes à l'aide d'ordinateurs puissants.
- Le premier à proposer le nombre hexadécimal à 64 chiffres ("hash") gagne le droit de former le nouveau bloc et de confirmer les transactions.
- Le mineur qui réussit est également récompensé par une quantité prédéterminée de crypto, connue sous le nom de "récompense de bloc".
- Comme il nécessite de grandes quantités de ressources de calcul et d'énergie pour générer de nouveaux blocs, les coûts d'exploitation derrière le PoW sont notoirement élevés.

PROOF OF WORK



Source : https://capital.com/proof-of-work-pow-definition

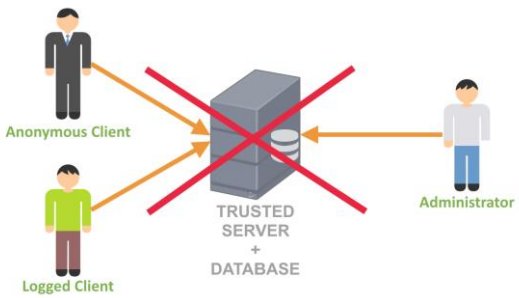
La Blockchain en tant que structure de données

1. Structure des Blockchains
2. Différents types de blockchain

La Blockchain en tant que structure de données

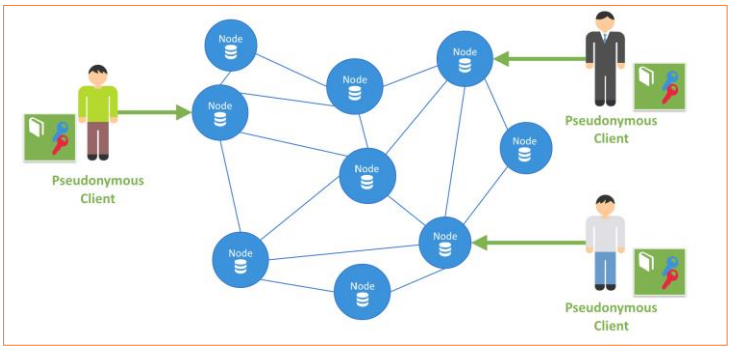
1. Structure des Blockchains

Plus besoin d'architectures client/serveur avec des rôles prédéfinis



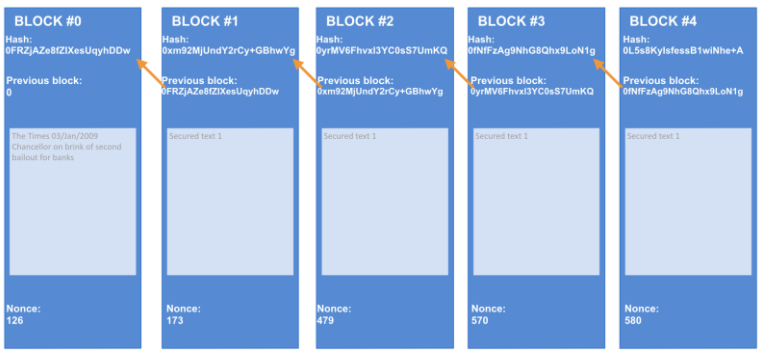
1. Structure des Blockchains

À la place, une architecture pair-à-pair, avec des paires de clé privées / publiques pseudonymes pour chaque client. Chaque nœud possède une copie du registre



1. Structure des Blockchains

Structure de données



La Blockchain en tant que structure de données

2. Différents types de blockchains

Il y a trois principaux types de Blockchain, qui ont émergés après l'apparition de Bitcoin.

- **Blockchain publique** :

Personne ne dirige le réseau, n'importe qui peut participer en lisant, écrivant, ou en auditant la blockchain (i.e. Bitcoin, Litecoin, etc.)

- **Blockchain privée** :

La propriété privée d'un individu ou d'une organisation, où une entité dirige l'écriture / la lecture ou bien les droits d'accès à l'écriture / à la lecture (i.e. Bankchain)

- **Blockchain fédérée / consortium** :

Plusieurs personnes dirigent le réseau. Un groupe d'entreprise ou d'individus représentant prennent des décisions communes au profit du réseau (i.e. r3, EWF)

DÉMO

https://andersbrownworth.com/blockchain/blockchain

SMART CONTRACTS

CONTRATS INTELLIGENTS

1. Théorie des contrats intelligents
   a. Théorie et architecture des contrats intelligents
   b. Architecture des systèmes autonomes et décentralisés
2. Applications des contrats intelligents

Applications blockchain existantes, structures liées et architectures

CONTRATS INTELLIGENTS

- Du bytecode stocké dans la blockchain
- Rédigé avec le langage Solidity
- Compilé
- Exécuté dans l'EVM (Ethereum Virtual Machine)
- Le bytecode est déployé sur Ethereum dans une transaction (dans le champs 'data' de la transaction)
- Une fois déposé dans la Blockchain, il est impossible de modifier ou de supprimer le contrat
- Il est accessible sur tous les nœuds à jour
- Pour interagir avec un contrat, il faut une machine virtuelle (EVM) qui interprète le bytecode
- EVM: Ethereum Virtual Machine est un environnement d'exécution des smart contrats

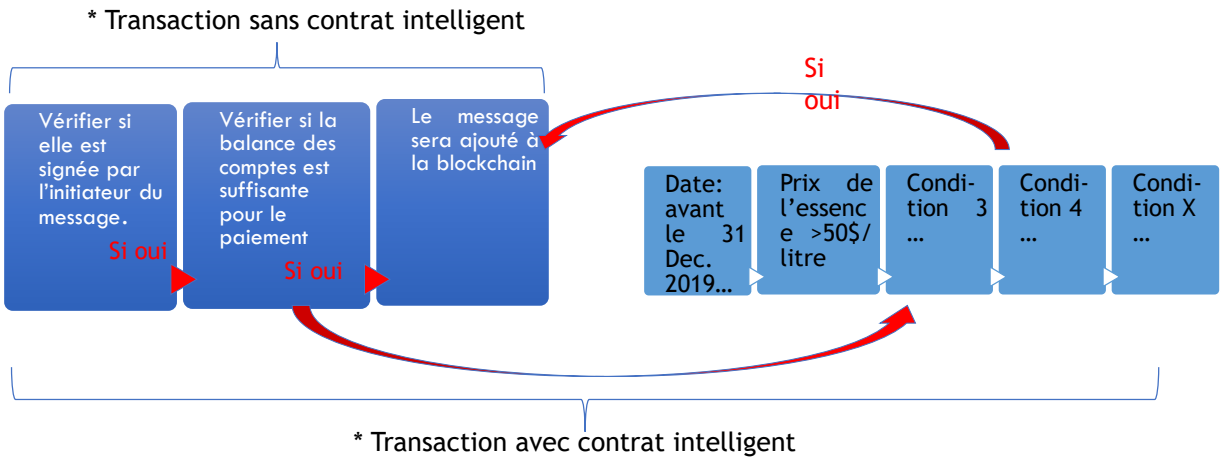
CONTRATS INTELLIGENTS

a. Théorie et architecture des contrats intelligents

- C'est un accord ou un ensemble de règles régissant une transaction entre entreprises.
- Il est mis en œuvre automatiquement dans le cadre d'une transaction.
- Les contrats intelligents peuvent comporter différentes clauses contractuelles susceptibles d'être, en tout ou partie, directement applicables et/ou exécutoires de manière automatique.
- Leur finalité est d'apporter une sécurité supérieure, mais aussi de diminuer les coûts et les délais par rapport aux contrats traditionnels.
- Il permet l'exécution fiable de transactions sans parties tierces de confiances

CONTRATS INTELLIGENTS

a. Architecture de contrats intelligents



CONTRATS INTELLIGENTS

Théorie et architecture des contrats intelligents

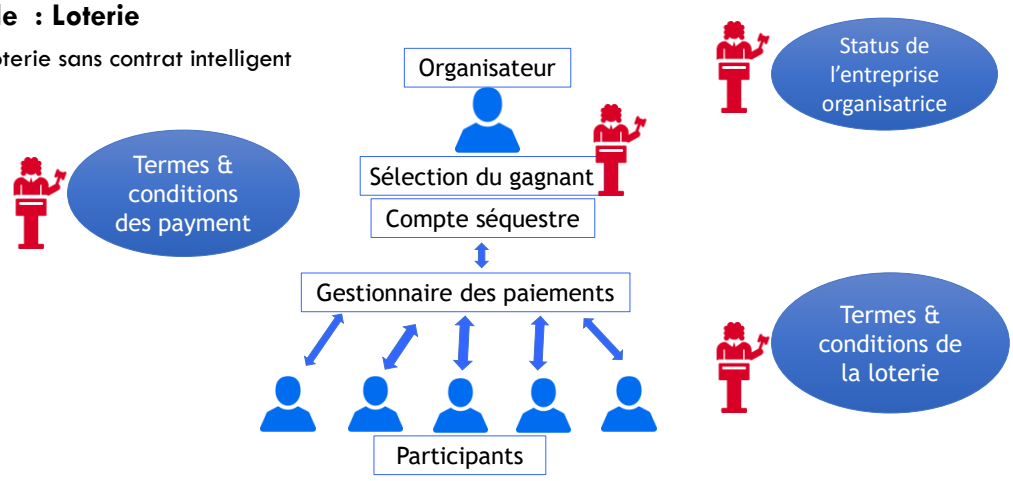
Exemples :

- Un contrat intelligent peut définir les conditions contractuelles régissant le transfert d'un titre d'obligation d'entreprise.
- Il peut aussi définir les conditions générales d'une assurance voyage, qui peuvent être automatiquement applicables lorsque, par exemple, un vol est retardé de plus de six heures.

CONTRATS INTELLIGENTS

Exemple : Loterie

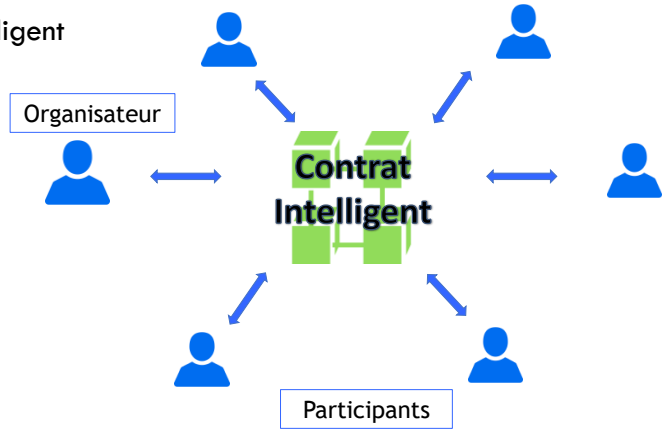
- Loterie sans contrat intelligent



CONTRATS INTELLIGENTS

Exemple : Loterie

Loterie avec contrat intelligent



CONTRATS INTELLIGENTS

CARACTÉRISTIQUES

1. **Distribué :** Tout le monde sur le réseau est assuré d'avoir une copie de toutes les conditions du contrat intelligent et elles ne peuvent pas être modifiées par l'une des parties.
2. **Déterministe :** les contrats intelligents ne peuvent exécuter les fonctions pour lesquelles ils sont conçus que lorsque les conditions requises sont remplies.
3. **Immuable :** Une fois le contrat intelligent déployé ne peut pas être modifié, il ne peut être supprimé que tant que la fonctionnalité est implémentée précédemment.
4. **Autonomie :** Aucun tiers n'est impliqué. Le contrat est conclu par vous et partagé entre les parties. Aucun intermédiaire n'est impliqué. De plus, le contrat intelligent est maintenu et exécuté par tous les nœuds du réseau.

CONTRATS INTELLIGENTS

CARACTÉRISTIQUES

5. **Personnalisable :** Les contrats intelligents ont la possibilité de modification, cad de personnalisation avant d'être lancés.
6. **Transparent :** les contrats intelligents sont toujours stockés sur un registre public distribué appelé blockchain grâce auquel le code est visible par tous, qu'ils soient ou non participants au contrat intelligent.
7. **Sans confiance :** Ceux-ci ne sont pas requis par des tiers pour vérifier l'intégrité du processus ou pour vérifier si les conditions requises sont remplies.
8. **Auto-vérification :** il s'agit d'une auto-vérification en raison de possibilités automatisées.
9. **Auto-exécutoire :** Ceux-ci sont auto-exécutoires lorsque les conditions et les règles sont remplies à toutes les étapes.

CONTRATS INTELLIGENTS

CAPACITÉS

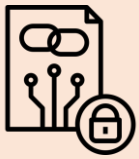
1. **Précision :** les contrats intelligents sont précis dans la limite où un programmeur les a codés avec précision pour l'exécution.
2. **Automatisation :** les contrats intelligents peuvent automatiser les tâches/processus effectués manuellement.
3. **Vitesse :** les contrats intelligents utilisent un code logiciel pour automatiser les tâches, réduisant ainsi le temps nécessaire pour manœuvrer à travers tous les processus liés à l'interaction humaine.
4. **Sauvegarde :** Chaque nœud de la blockchain maintient le grand livre partagé, fournissant probablement la meilleure fonction de sauvegarde.

CONTRATS INTELLIGENTS

CAPACITÉS

5. **Sécurité :** la cryptographie peut garantir que les actifs sont sains et saufs. Même si quelqu'un casse le cryptage, le pirate devra modifier tous les blocs qui viennent après le bloc qui a été modifié.
6. **Économies :** les contrats intelligents permettent d'économiser de l'argent car ils éliminent la présence d'intermédiaires dans le processus.
7. **Gère les informations :** le contrat intelligent gère l'accord des utilisateurs et stocke des informations sur une application, telles que l'enregistrement de domaine, les enregistrements d'adhésion, etc.
8. **Comptes multi-signatures :** les contrats intelligents prennent en charge les comptes multi-signatures pour distribuer les fonds dès que toutes les parties concernées confirment l'accord.

SMART CONTACT



APPLICATION I : Smart contact sous remix IDE

i. Premier smart contract
ii. Compiler un smart cantract
iii. Déployer le smart contract
iv. Tester le smart contract

SOLIDITY

LANGAGE DE PROGRAMMATION D'ETHEREUM

SOLIDITY

- Solidity est le langage de programmation de la plateforme d'applications décentralisées la plus connue et la plus utilisée : Ethereum.
- Solidity permet de coder des smart contracts – un ensemble d'instructions qui permettent au réseau de mettre à jour l'état de la blockchain en fonction des conditions et des variables définies par leur créateur.

SOLIDITY

- Un langage de programmation orienté objet qui présente des similitudes avec JavaScript ou C++.
- Un langage de haut niveau, qui sera compilé en langage de bas niveau (*bytecode*) pour être interprété par l'environnement d'exécution d'Ethereum.
- Il s'agit de **la EVM** : environnement d'exécution **isolé du réseau** et permet d'interpréter le code Solidity une fois compilé, afin de garantir la mise à jour de l'état des contrats via tous les nœuds validateurs du réseau.
- Ce sont les mineurs qui exécutent ces instructions et mettent à jour la blockchain d'Ethereum.
- Chaque instruction d'un contrat Solidity aura donc **un coût d'exécution** (en *gas*).

→ *Afin d'éviter les boucles infinies au sein d'un programme, il y a toujours une limite au **gas** qui pourra être consommé.*

SOLIDITY

- Basé sur les **contrats Solidity** (équivalent au classes) sont définis par :
  - des variables d'état
  - des fonctions
  - des modificateurs
  - des événements
  - des types.
- Il est possible d'appeler d'autres contrats à l'intérieur d'un contrat (message calls),
- Il est possible d'injecter des données externes au sein d'un contrat
- Il existe aussi des contrats spéciaux : les librairies et les interfaces.

SOLIDITY

L'EVM dispose de trois zones distinctes pour traiter les données :

- **La zone de stockage :** associée à un compte Ethereum, elle est persistante et intervient entre l'appel d'une fonction et une transaction.
- **La zone de mémoire :** associée à chaque contrat et mise à jour lors de chaque appel (*message call*), sa taille varie proportionnellement à la quantité de données gérées par le smart contract.
- **La pile** de la machine virtuelle : elle comporte les instructions du contrat, qui seront exécutées

SOLIDITY

Les interfaces

- Spécifient la structure des fonctions d'un contrat (nom, entrées/sorties) sans décrire le code.
- Elles sont utiles pour interagir avec plusieurs **contrats externes** et créer des contrats complexes, comme en faisant appel à des librairies.

Les librairies

- Semblables à des contrats déployés indéfiniment sur la blockchain, dont le code est réutilisé par de multiples contrats y faisant appel.
- Les librairies sont donc du code isolé, qui sera exécuté spécifiquement pour le smart contract les intégrant.

SOLIDITY

Environnements de développement

- **Remix** est l'IDE la plus utilisée,
- Gett+ Truffle + Ganache + Atom (ou bien VS Code/Intellij)
- Solidity Extension for Visual Studio + Blockapps-bloc
- Hardhat
- Et bien d'autres

SOLIDITY

Variables :

Variable d'état:

- les valeurs de ces variables sont stockées en permanence dans le stockage des contrats.
- Chaque fonction a sa propre portée et les variables d'état doivent toujours être définies en dehors de cette portée.
- Exemple :

le *contrat Solidity_var_Test* initialise les valeurs de la variable d'état state_var entière non signée à l'aide d'un constructeur.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;
contract Solidity_var_Test {
     // Déclaration d'une variable d'état
   uint8 public state_var;
     // Définition du constructeur
   constructor(){
      state_var = 16;
   }
}
```


SOLIDITY

Variables :

Variable locale:

- les valeurs de ces variables sont présentes jusqu'à ce que la fonction s'exécute et elles ne sont pas accessibles en dehors de cette fonction.
- Ce type de variable est généralement utilisé pour stocker des valeurs temporaires.
- Exemple :

Fonction pour déclarer et initialiser les variables locales et renvoyer la somme des deux variables locales

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;
contract Solidity_var_Test {
     // Définition d'une fonction
   function getResult() public pure returns(uint){
     // Initialisation des variables locales
      uint local_var1 = 1;
      uint local_var2 = 2;
      uint result = local_var1 + local_var2;
      // Accès au variable locale
      return result;
   }
}
```

SOLIDITY

Variables :

Variable globale:

- il s'agit de variables spéciales qui peuvent être utilisées globalement et donnent des informations sur les transactions et les propriétés blockChain.
- Certaines des variables globales sont listées ci-dessous :
- Exemple :

le *contact Test* utilise la variable *msg.sender* pour accéder à l'adresse de la personne déployant le contrat.

```
pragma solidity ^0.8.9;
contract Test {
  // Définition d'une variable
  address public admin;
  // Création d'un constructeur
  constructor(){
    admin= msg.sender;
  }
}
```


SOLIDITY

Types de bases :

1. Types de taille fixe

uint : nombres entier positifs

bool

adress

bytes32

2. Types de taille dynamique

string

bytes

uint [ ]

Mapping(uint => string) users

3. Types définit par le programmeur

```
struct user{
    uint id;
    string nom;
    uint[] certifIds;
}
```

address

- address contient la valeur de 20 octets représentant la taille d'une adresse Ethereum.
- Une adresse peut être utilisée pour obtenir le solde en utilisant la méthode .balance et peut être utilisée pour transférer le solde vers une autre adresse en utilisant la méthode .transfer.

```
address x = 0x212;
address myAddress = this;
if (x.balance > 10 && myAddress.balance <= 10)
    x.transfer(10);
```


Address - Exemple

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0ntract SimpleTransfer {
    // Fonction pour recevoir de l'Ether
    receive() external payable {}
    // Fonction pour envoyer une somme à une adresse, avec vérification du solde
    function transferTo(address payable recipient, uint amount) public {
        require(address(this).balance >= amount, "Solde insuffisant dans le contrat");
        recipient.transfer(amount);
    }
    // Affiche le solde du contrat
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}
```

Portée des variables

- La portée des variables locales est limitée à la fonction dans laquelle elles sont définies
- les variables d'état peuvent avoir trois types de portée.
  - Public : sont accessibles en interne et par le biais de messages. Pour une variable d'état publique, une fonction getter automatique est générée.
  - Internal : sont accessibles qu'en interne, à partir du contrat en cours ou du contrat qui en dérive sans utiliser cette fonction.
  - Private : ne sont accessibles qu'en interne, à partir du contrat en cours ou du contrat qui en dérive.

Portée des variables

```
pragma solidity ^0.5.0;
contract C {
   uint public data = 30;
   uint internal iData= 10;
   uint public storedData;
   function x() public returns (uint) {
      data = 3; // internal access
      return data;
   }
}
contract Caller {
   C c = new C();
   function f() public view returns (uint)
       {
      return c.data(); //external access
       }
}
```

```
contract D is C {
  function y() public returns (uint)
   {
      iData = 3; // internal access
      return iData;
   }
  function getResult() public returns(uint)
    {
      uint a = 1; // local variable
      uint b = 2;
     //access the state variable
      storedData = a + b;
      return storedData;
    }
}
```

Modificateurs de fonction

- Les modificateurs de fonction sont utilisés pour modifier le comportement d'une fonction. Par exemple pour ajouter une condition préalable à une fonction.
- Le corps de la fonction est inséré là où le symbole spécial "_ ;" apparaît dans la définition d'un modificateur.
- Si la condition du modificateur est satisfaite lors de l'appel de cette fonction, la fonction est exécutée et sinon, une exception est levée.

Modificateurs de fonction

Exemple :

```
pragma solidity ^0.5.0;
contract Proprietaire {
    address prop;
    constructor() public {
        prop = msg.sender;
    }
    modifier onlyPropietaire {
        require(msg.sender == prop);
        _;
    }
    modifier couts(uint prix) {
        if (msg.value >= prix) {
            _;
        }
    }
}
```

```
contract Register is Proprietaire {
    mapping (address => bool) registeredAddresses;
    uint prix;
    constructor(uint prixInitial) public {
         prix = prixInitial; }

    function register() public payable couts(prix) {
        registeredAddresses[msg.sender] = true;
    }
    function changePrix(uint _prix)public onlyPropietaire
         {
           prix = _prix;
         }
}
```

View Fonctions :

Les View fonctions garantissent qu'elles ne modifieront pas l'état du contrat.

```
contract Test {
   uint sum;
   function getResult() public view returns(uint product, uint sum)
  {
      uint a = 1; // local variable
      uint b = 2;
      uint product = a * b;
      uint sum = a + b; //Erreur
   }
}
```


Fonctions pures :

- Les fonctions "pure" garantissent qu'elles ne lisent ni ne modifient l'état du contrat
- Les fonctions "pure" peuvent utiliser les fonctions revert() et require() pour annuler les changements d'état potentiels si une erreur se produit.

```
pragma solidity ^0.5.0;
contract Test {
    uint x=4;
   function getResult() public pure returns(uint product){
      uint y = 10;
      product = x * y; //Erreur
   }
}
```

SOLIDITY

Function Overloading

- Plusieurs définitions pour le même nom de fonction dans la même portée.
- Les définitions de la fonction doivent différer les unes des autres par les types et/ou le nombre d'arguments dans la liste des arguments.

```
contract Test {
   function CalculSomme(uint a, uint b) public pure returns(uint)
       {
              return a + b;
       }
   function CalculSomme(uint a, uint b, uint c) public pure returns(uint)
       {
              return a + b + c;
       }
}
```


SOLIDITY

Mathematical Functions

- Solidity fournit également des fonctions mathématiques intégrées.
- Voici les méthodes les plus utilisées

```
addmod(uint x, uint y, uint k) returns (uint); // calcule (x + y) % k

mulmod(uint x, uint y, uint k) returns (uint); // calcule (x * y) % k
```

Storage & memory

memory : emplacement temporaire pour stocker des données;

stockage : emplacement contenant des données entre les appels de fonction

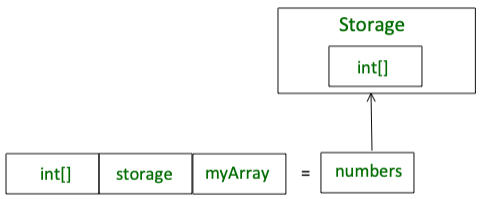
- Les variables d'état et les variables locales des structs, array sont toujours stockées dans le stockage par défaut.
- Les arguments de la fonction sont en mémoire.
- Chaque fois qu'une nouvelle instance d'un tableau est créée à l'aide du mot clé 'memory', une nouvelle copie de cette variable est créée. La modification de la valeur de tableau de la nouvelle instance n'affecte pas le tableau d'origine.

Storage & memory

Exemple : storage

```
pragma solidity ^0.4.17;
// Creation d'un contrat
contract helloGeeks
{
// déclaration d'un tableau
int[] public numbers;
// Fonction pour remplir le tableau
function Numbers() public
{
    numbers.push(1);
    numbers.push(2);
    //Creatition d'une nouvelle instance
    int[] storage myArray = numbers;
    // affecter une nouvelle valeur au nouveau tableau
    myArray[0] = 0;
}
}
```

Contenu du tableau « numbers » : **[0,2]**

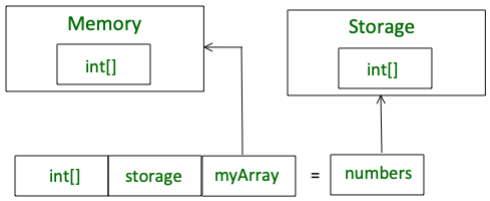


Storage & memory

Exemple : memory

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.4.17;
// Creation d'un contrat
contract helloGeeks
{
// déclaration d'un tableau
int[] public numbers;
// Fonction pour remplir le tableau
function Numbers() public
{
    numbers.push(1);
    numbers.push(2);
    //Creatition d'une nouvelle instance
    int[] memory myArray = numbers;
    // affecter une nouvelle valeur au nouveau tableau
    myArray[0] = 0;
}
}
```

Contenu du tableau « numbers » : **[1,2]**



Contrat - Quantificateurs de visibilité

Différents quantificateurs de visibilité pour les fonctions/variables d'état d'un contrat:

- **external** : Les fonctions externes sont destinées à être appelées par d'autres contrats.
  - Elles ne peuvent pas être utilisées pour un appel interne. Pour appeler une fonction externe dans un contrat, il faut appeler this.function_name().
  - Les variables d'état ne peuvent pas être marquées comme externes.
- **public** : Les fonctions/variables publiques peuvent être utilisées à la fois en externe et en interne. Pour une variable d'état publique, Solidity crée automatiquement une fonction getter.
- **Internal** : Les fonctions/variables internes ne peuvent être utilisées qu'en interne ou par des contrats dérivés.
- **private** : Les fonctions/variables privées ne peuvent être utilisées qu'en interne et pas même par des contrats dérivés.

SOLIDITY

Héritage

- L'héritage est un moyen d'étendre la fonctionnalité d'un contrat.
- Solidity prend en charge l'héritage simple et l'héritage multiple. Voici les principaux points forts.
- Un contrat dérivé peut accéder à tous les membres non privés, y compris les méthodes internes et les variables d'état. Mais leur utilisation n'est pas autorisée.
- La substitution de fonction est autorisée à condition que la signature de la fonction reste la même. En cas de différence entre les paramètres de sortie, la compilation échouera.
- On peut appeler la fonction d'un super contrat en utilisant le mot clé super ou le nom du super contrat.
- En cas d'héritage multiple, l'appel de la fonction à l'aide de super donne la préférence au contrat le plus dérivé.

SOLIDITY

Constructeur

- Le constructeur est une fonction spéciale déclarée à l'aide du mot clé constructor.
- Il s'agit d'une fonction facultative utilisée pour initialiser les variables d'état d'un contrat.
- Un contrat ne peut avoir qu'un seul constructeur.
- Un code constructeur est exécuté une fois lorsqu'un contrat est créé et il est utilisé pour initialiser l'état du contrat.
- Après l'exécution d'un code constructeur, le code final est déployé sur la blockchain. Ce code comprend les fonctions publiques et le code accessible par les fonctions publiques. Le code du constructeur ou toute méthode interne utilisée uniquement par le constructeur ne sont pas inclus dans le code final.
- Un constructeur peut être soit public, soit interne.
- Un constructeur interne marque le contrat comme abstrait.
- Dans le cas où aucun constructeur n'est défini, un constructeur par défaut est présent dans le contrat.

SOLIDITY

Contrat abstract

- Un contrat abstrait est un contrat qui contient au moins une fonction sans aucune implémentation.
- Un tel contrat est utilisé comme contrat de base. En général, un contrat abstrait contient à la fois des fonctions implémentées et des fonctions abstraites.
- Le contrat dérivé implémentera la fonction abstraite et utilisera les fonctions existantes si nécessaire.
- Dans le cas où un contrat dérivé n'implémente pas la fonction abstraite, ce contrat dérivé sera marqué comme abstrait.

SOLIDITY

Contrat abstract

- Exemple :

```
contract Calculator {
   function getResult() public view returns(uint);
}
contract Test is Calculator {
   function getResult() public view returns(uint) {
      uint a = 1;
      uint b = 2;
      uint result = a + b;
      return result;
   }
}
```

Interface

- Les interfaces sont similaires aux contrats abstraits et sont créées à l'aide du mot clé interface.
- L'interface ne peut pas avoir de fonction avec implémentation.
- Les fonctions d'une interface ne peuvent être que de type externe.
- L'interface ne peut pas avoir de constructeur.
- L'interface ne peut pas avoir de variables d'état.
- L'interface peut avoir des enum, des structs auxquels on peut accéder en utilisant la notation par points du nom de l'interface.

SOLIDITY

Interface

- Les interfaces sont similaires aux contrats abstraits et sont créées à l'aide du mot clé interface.
- L'interface ne peut pas avoir de fonction avec implémentation.
- Les fonctions d'une interface ne peuvent être que de type externe.
- L'interface ne peut pas avoir de constructeur.
- L'interface ne peut pas avoir de variables d'état.
- L'interface peut avoir des enum, des structs auxquels on peut accéder en utilisant la notation par points du nom de l'interface.

Library

- Les bibliothèques sont similaires aux contrats mais sont principalement destinées à être réutilisées.
- Une bibliothèque contient des fonctions que d'autres contrats peuvent appeler.
- Solidity impose certaines restrictions quant à l'utilisation d'une bibliothèque.
- Les fonctions de la bibliothèque peuvent être appelées directement si elles ne modifient pas l'état.
- Cela signifie que seules les fonctions pures ou view peuvent être appelées depuis l'extérieur de la bibliothèque.
- La bibliothèque ne peut pas être détruite car elle est supposée être sans état.
- Une bibliothèque ne peut pas avoir de variables d'état.
- Une bibliothèque ne peut hériter d'aucun élément.
- Une bibliothèque ne peut pas être héritée

Library - Exemple

```
library Search {
   function indexOf(uint[] storage self, uint value) public view returns (uint) {
      for (uint i = 0; i < self.length; i++) if (self[i] == value) return i;
      return uint(-1);
  }}
contract Test {
   uint[] data;
   constructor() public {
      data.push(1); data.push(2); data.push(3);
      data.push(4); data.push(5);
   }
   function isValuePresent() external view returns(uint){
      uint value = 4;
      //search if value is present in the array using Library function
      uint index = Search.indexOf(data, value);
      return index;
}}
```

Event

- L'événement est un membre héritable d'un contrat.
- Un événement est émis, il stocke les arguments passés dans les journaux de transactions.
- Ces journaux sont stockés sur la blockchain et sont accessibles en utilisant l'adresse du contrat jusqu'à ce que le contrat soit présent sur la blockchain.
- Un événement généré n'est pas accessible depuis les contrats, pas même ceux qui l'ont créé et émis.
- Un événement peut être déclaré en utilisant le mot clé event.

Error Handling

Solidity fournit diverses fonctions pour la gestion des erreurs. En général, lorsqu'une erreur se produit, l'état est rétabli à son état initial. D'autres contrôles visent à empêcher l'accès au code non autorisé. Voici quelques-unes des principales méthodes utilisées pour la gestion des erreurs

- assert(bool condition) : Dans le cas où la condition n'est pas remplie, cet appel de méthode provoque un opcode invalide et toute modification apportée à l'état est annulée. Cette méthode doit être utilisée pour les erreurs internes.
- require(bool condition) : Si la condition n'est pas remplie, l'appel de cette méthode ramène l'état original. Cette méthode doit être utilisée pour les erreurs dans les entrées ou les composants externes.

Error Handling

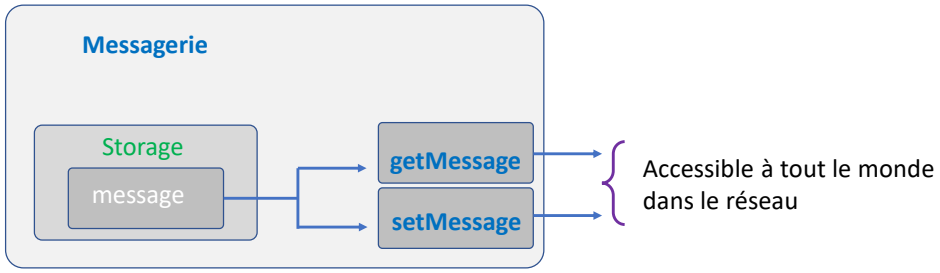
- require(bool condition, string memory message) : Si la condition n'est pas remplie, l'appel de cette méthode revient à l'état initial. Cette méthode doit être utilisée pour les erreurs dans les entrées ou les composants externes. Elle offre la possibilité de fournir un message personnalisé.
- revert() : Cette méthode interrompt l'exécution et rétablit tous les changements apportés à l'état.
- revert(string memory reason) : Cette méthode interrompt l'exécution et annule tout changement apporté à l'état. Elle offre la possibilité de fournir un message personnalisé.

Exemple de contact

```
contract Messagerie {
    string public message;
        constructor(string initialMessage) public {
        message = initialMessage;
    }
    function setMessage(string newMessage) public {
        message = newMessage;
    }
    function getMessage() public view returns (string) {
        return message;
    }
}
```

DÉPLOIEMENT SOUS REMIX IDE

Exemple de contact

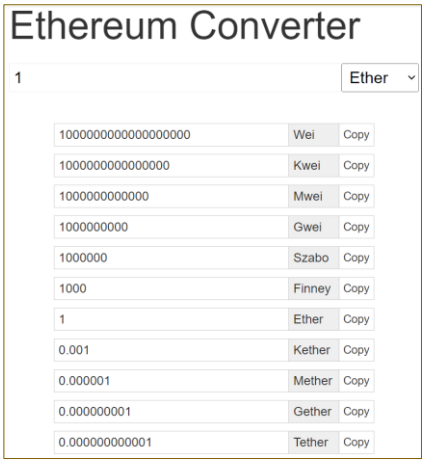


EXÉCUTION DES FONCTIONS DU CONTRAT

Appel d'une fonction	Envoie d'une transaction à une fonction
On peut pas modifier les données du contrat	On peut modifier les données du contrat
Peut retourner des données	Retourne le Hash de la transaction
Lancée instantanément	Nécessite un temps d'exécution
Libre de faire	Coût

WEI VS ETHER

https://converter.murkin.me/



GAZ ET TRANSACTIONS

```
function doCalcul(int a,int b){
        a+b;
        b-a;
        a*b;
        a==0;
    }
```

Transaction appelant la fonction doClacul
gazPrice : 300
gazLimit : 10

Coût de la fonction doCalcul	
ADD	3 Gaz
SUBSTRACT	3 Gaz
MULTIPLY	5 Gaz
EQ	3 Gaz

SMART CONTRAT AVEC SOLIDITY

APPLICATION 2 : Installation de l'environnement de développement


Geth


Ganache


TRUFFLE

LES OUTILS A INSTALLER

Geth (Go-Ethereum)

Une l'interface de ligne de commande pour l'un des trois principaux composants de code de la blockchain Ethereum .

Lien : https://geth.ethereum.org/downloads/

Puis ajouter Geth aux variables d'environement

geth version

LES OUTILS A INSTALLER

Ganache

Un simulateur Ethereum qui rend le développement d'applications Ethereum plus rapide, plus facile et plus sûr. Il inclut toutes les fonctions et fonctionnalités RPC populaires et peut être exécuté de manière déterministe pour faciliter le développement.

https://trufflesuite.com/ganache/

LES OUTILS A INSTALLER

Node

https://nodejs.org/

- node –v
- npm –v : un gestionnaire de packages pour les packages Node.js

LES OUTILS A INSTALLER

Truffle:

- Truffle est le framework de développement le plus populaire pour Ethereum
- S'occupe de gérer vos artefacts de contrat afin que vous n'ayez pas à le faire.
- Inclut la prise en charge des déploiements personnalisés, des liaisons de bibliothèques et des applications Ethereum complexes.

npm uninstall -g truffle

npm install -g truffle@4.1.15

Mettre à jour : npm i -g truffle@4.1.15

LES OUTILS A INSTALLER

Atom ou bien VS Code

https://github.com/atom/atom/releases

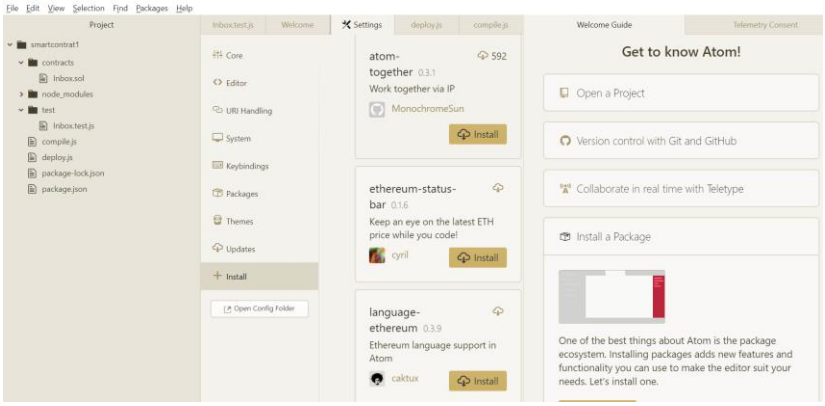
atom -v

apm -v

LES OUTILS A INSTALLER

Ethereum

apm install language-ethereum



LES OUTILS A INSTALLER

Git

Un système de contrôle de version distribué gratuit et open source conçu pour tout gérer, des petits aux très grands projets, avec rapidité et efficacité.

https://git-scm.com/download/win/

SMART CONTACT

APPLICATION 2

INTERAGIR AVEC LE SMART CONTRACT

i. Créer un nouveau projet

ii. Ajouter un smart contrat

iii. Déployer le smart contrat

- Sous le réseau « developpment »
- Sous Ganache

iv. Interagir avec le smart contract

SMART CONTACT

APPLICATION 3

CONFIGURATION D'UN NŒUD PRIVÉ

i. Créer un nouveau projet

ii. Créer un genesis bloc

iii. Initialiser un nœud privé

iv. Créer des comptes

v. Effectuer des transactions