Le langage solidity

Demo Remix

- http://remix.ethereum.org

Les variables et leur assignation par défaut.

- Les entiers non signés ou non (INT / UINT) (de 8 a 256 bits) initialisés a 0 (exemple uint8)
- Les strings initialisées a vide
- Les booleans initialisés a false.
- Address initialisés a 0x0000000000000000000000000000000000000000 /!\ envoyer quoi que ce soit à cette adresse sera irrécupérable, elle est connue comme l'adresse de burn (dans tous les cas envoyer à un adresse mal tapée = perte des fonds)
- Il existe d'autres types de variables tels que fixed un fixed pour des nombres décimaux (rarement utilisés)

Les types de variables

- Des variables peuvent avoir plusieurs sources :
  - La variable d’état (state variable) qui sont stockées en permanence dans le contrat
  - La variable locale qui n’existe que le temps d’execution d’une fonction
  - La variable globale qui sont des variables de la blockchain (exemple block.timestamp)

Les variables globales

Name	Returns
blockhash(uint blockNumber) returns (bytes32)	Hash of the given block - only works for 256 most recent, excluding current, blocks
block.coinbase (address payable)	Current block miner's address
block.difficulty (uint)	Current block difficulty
block.gaslimit (uint)	Current block gaslimit
block.number (uint)	Current block number
block.timestamp (uint)	Current block timestamp as seconds since unix epoch
gasleft() returns (uint256)	Remaining gas
msg.data (bytes calldata)	Complete calldata
msg.sender (address payable)	Sender of the message (current caller)
msg.sig (bytes4)	First four bytes of the calldata (function identifier)
msg.value (uint)	Number of wei sent with the message
now (uint)	Current block timestamp
tx.gasprice (uint)	Gas price of the transaction
tx.origin (address payable)	Sender of the transaction

La portée des variables d’état

- Private : statut d’une variable par défaut, ne sont accessibles que dans le contrat. Pour la modifier ou la consulter il faudra utiliser une fonction. (Compiler contrat VariablePortée), peuvent être accédées par des contrats enfants.
- Public : un getter est crée directement au niveau du contrat elle sera donc consultable directement de l’exterieur du contrat.
- Internal : peuvent être accédé seulement depuis le contrat.

Portée des fonctions

- Les fonctions private : uniquement utilisables dans le contrat
- Les fonctions internal : comme private mais accessible depuis un contrat qui hérite de celui ci.
- Les fonctions external : utilisable en externe exclusivement
- Les fonctions public : tout le monde peut les autoriser.

Les opérateurs

- Bool : true ou false
- ! Negation
- && and
- || ou
- == égalité
- != different de
- <= et < inférieur ou égal et inférieur
- >= et > supérieur ou égal et supérieur
-

Addition soustraction division ...

- + - / * %
- ** exponentiation

Helloworld

- Le contrat est composé d’une variable d’état, et d’une fonction.
- La fonction retourne une string (nous verrons pourquoi mémory plus tard)
- Et le return qu’il compose.
- Compiler et déployer pour tester.

Contrat 2_numbers

- Ici nous allons jouer avec une variable d’état.
- La fonction update toto permet de mettre à jours le nombre.

Exercice

- Créer un contrat qui récupère l’horodatage du block actuel

Contrat 3_increment

- Qui montre comment incrémenter et décrémenter une variable (identique à beaucoup de langages).

Contrat 4_owner

- La fonction constructor est la fonction appelle au déploiement du contrat;
- Nous pouvons passer des paramètres en entrée.
- Dans le cas présent nous récuperons la variable msg.sender qui est la personne qui crée le contrat.
- Msg.sender est toujours la personne qui interagit avec le contrat.

5 secure_owner

- Le contrat change owner n’a pas de sécurité pour s’assurer que personne d’autre ne peut modifier le owner.
- Dans ce cas nous utilisons Require(msg.sender == owner, « texte »);
- Si require n’est pas vérifié la fonction renvoi une erreur.

Whitelist

- Le contrat whitelist prend en charge un nouvel élément le mapping
- Le mapping associé clé valeur.
- Dans notre cas le mapping whitelist donne un boolean pour une adresse.
- Donc l’adresse sera autorisée ou refusée à faire qq chose.
- Nous avons également 2 éléments interessants : le event et le émit.
- Event permet de définir un evemenent que nous souhaitons mettre en avant.
- Emit appellera l’event avec le paramètre associé. (Ici le émit attends une adresse en entrée et enverra le message que l’adresse à été autorisée.

Whitelist exercice

- S’assurer que seul le owner puisse définir une adresse dans la whitelist
- Exercice bonus :
  - Faire en sorte qu’une adresse autorisée puisse être refusée.

Whitelist avancées :

- Défini une struct (structure de donnée expliquée plus loins)
- Par convention on nomme un struct avec une majuscule et lorsque l'on instancie une struct son nom sera en minuscule.
- Exemple d'initialisation :

```
Person memory person; *// Déclaration d'une variable de type Person*
person.name = "name"; *// Initialisation du premier paramètre de la struct*
person.age = 30; *// Initialisation du deuxième paramètre de la struct*
```

- Explication du reste dans le code.

Les struct

- Les structs sont des schema de structure données. (Comme un objet en JS)
-

```
struct User {
    string name;
    uint8 age;
}
```

Les modifier(s)

- Un modifier créer une option de fonction qui vérifiera quelque chose. Cela évite les taches répétitives liées au require par exemple.
- Un modifier commence toujours par son mot clé et se termine par _;
-

Exercice

- Sur le contrat whitelist avancée remplacer les require de chaque fonction par un modifier

Les boucles

- Le for : for (uint I = 0; loops.lenght>100;i++) {} relativement équivalent au js
- Le while : while (I !=0) {}
- Do while : do {I = 10} while (I >100) {}

Les if

- Exemple

```
function test() public view returns (uint256) {
        if (valeur < 5) {
            return 0;
        } else if (valeur < 10) {
            return 1;
        } else {
            return 2;
        }
    }
```

-

Les tableaux

- Les fonctions tableaux en solidity sont pauvres :
  - Push
  - Length
  - Pop
  - Donc pour supprimer une valeur a un index donné utiliser l’exemple donné dans deleteByIndex.

Exercice

- Vous devez créer un contrat de gestion des notes des élèves.
- Vous devez faire un mapping de note des élèves
- Une fonction doit retourner la moyenne d’un élève par son adresse.Seuls les professeurs peuvent ajouter des notes
- Il est possible de changer une note par son index
-

Les enums

- Les enums sont un type de variables avec un nombre restreint de valeurs que nous pouvons créer nous meme.
- Déclaration : enum State {Created, Locked, Inactive} //pas de ; a la fin
- Creation de l’enum :
  - State public defaultstate
  - Defaultstate = state.Created;

Les fonctions payable

- Il existe en solidity plusieurs moyens de recevoir des éthers
- La fonction receive
- Et une fonction de dépot qui doit être identifiée comme payable.
- Dans les 2 cas ces deux fonctions utiliserons 2 paramètres le msg.sender et le msg.data
- Msg.data contiendra la valeurs des éthers envoyés valeur en wei.

Assert revert & require traitement exceptions

- Require est la pour verifier une conditions en général au début d'une fonction
- Assert syntaxe identique mais plus utilisée pour vérifier le résultat dans une fonction
- Dans les faits assert et require sont plutôt des jumeaux
- Revert() est appelle pour stopper une exécution dans une fonction.
- Personnellement j'utiliser beaucoup require, et si je dois faire des vérifications j'utilise un if avec un revert.(je ne sais pas si je fais bien ;p)

Heritage entre contrats

Les fonctions fallback

- La fonction fallback est appellée uniquement dans le cas d'appel d'une fonction qui n'existe pas. Elle peut également être appellée si la fonction receive() pour recevoir des éthers est appellée et qu'elle n'est pas présente.
- Son cout minimal en gas est de 2300
- Il ne doit y avoir qu'une seule fonction fallback par contrat.
-

Utiliser les tokens ERC20 dans les contrats

- Introduction aux interfaces :
  - Les interfaces sont des définitions de fonctions qui permettent d'instance des contrats déja existants. Elles reprennent les fonctions que l'on souhaite utiliser lors de la communication avec les contrats. On place toujours la définition de l'interface dans le fichier solidity avant le contrat.

Créer un contrat ERC20

- Les contrats ERC20 sont définis selon le standard de l'ethereum consortium : https://ethereum.org/fr/developers/docs/standards/tokens/erc-20/
- Name : Ils ont un nom
- Un symbole
- Un nombre de décimales
- La totalsupply
- La balance de l'utilisateur (balanceOf)
- Une fonction de transfert
- Une fonction transferFrom qui permet a une autre adresse d'envoyer ces tokens a une autre Addresse
- Une fonction approve qui permet d'autoriser une adresse a envoyer des tokens d'une adresse a une autre. (S'utilise de paire avec transferFrom)
- Une fonction allowance est le montant restant des tokens que l'adresse approuvée peut encore envoyer.
- Toutes les valeurs sont en wei.

Créer un contrat ERC20 partie 2

- Utiliser openzepplin : https://docs.openzeppelin.com/contracts/3.x/erc20#:~:text=By%20default%2C%20ERC20%20uses%20a,call%20_setupDecimals%20in%20your%20constructor.

- Deployer avec remix

-

Exercice ERC20

- Modifier le contrat 20_ERC20 pour que lorsque l’on appelle le constructor, la maxSupply soit juste un point de référence à ne pas dépasser.
- Ajouter une fonction mint qui pourra minter les tokens jusqu’a arriver a la maxSupply de 100m de tokens.
-

Exercice expliquez moi ces contrats ERC20 :

WETH :
https://etherscan.io/token/0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2#code

EURCV:
https://etherscan.io/token/0xf7790914Dc335B20Aa19D7c9C9171e14e278A134

Nextep :
https://bscscan.com/address/0xf10770649b0b8f62bb5e87ad0da7729888a7f5c3#code

correction

- WETH est un contrat qui permet de convertir des éthers en weth a 1:1 pour obtenir un ERC20 au lieu d'un token de blockchain. (Souvent utilisé dans la defi)
- EURCV : contrat de la société générale. Ce contrat, attend que la banque valide les transactions …, possibilité pour la banque de saisir les tokens depuis votre compte… entre autres :)
- Nextep : contrat qui peut être bloqué sauf par les admins pour pouvoir vendre les tokens en cas de crash et sauver les meubles de leur argent a eux…

ERC721 Les NFT

- Les contrats types ERC721 est un standard pour les tokens non fongible.
- La non fongibilité permet de représenter des actifs uniques tels que les oeuvres d’arts objets de collections ou droits de propriété.
- Les tokens sont stockés dans des contrats intelligents et peuvent être transferés entre utilisateurs.
- Chaque token est identifiable par un id unique ce qui le rend non interchangeable.

ERC721 que contiennent les contrats

- La fonction principale sera le mint (création du token)
- Le transfert entre utilisateurs
- Les fonctions transferFrom & approve comme dans l'erc20
- Ils peuvent comporter d'autres spécificités comme des métadonnées stockés onchain ou offchain (ipfs).

ERC721 BoredApes visu du contrat

- Dispo a cette adresse : https://etherscan.io/address/0xbc4ca0eda7647a8ab7c2061c2e118a18a936f13d#code
-

TP noté sur 2h

- 1 création d’un système de licenses a durée déterminée.
- Vous devez créer un système de vente de license logiciels à base de NFT.
- Chaque license sera unique et elle posséderons un timestamp de début et de fin.
- 3 fonctions : Achat pour un mois achat pour 3 mois Achat pour un an.
- L’achat se fera en éther.
- L’admin du contrat pourra définir des opérateurs qui pourront minter des nfts sans paiement (car système de paiement externe au contrat.
- Nous pouvons retirer le montant de la balance.
- Comme les licenses ont étés développées par plusieurs entreprises nous devrons utiliser le split payment de openzepplin pour que l’argent soit réparti entre les différents payeurs.
- A la fin le NFT timestampé doit se retrouver sur l’adresse de l’utilisateur.
- La sécurité est la priorité notée sur 10 devoir noté sur 20. Si une fonction est accessible par quelqu’un qui ne devrait pas dans ce cas j’enlève 2 points.
- Droit a internet par contre interdiction pour chatgpt ou tout autre IA générant du code;)

TP2 si vous êtes trop forts ;)

- Créer un contrat de gestions de billetteries a base de NFT.
- Les billets sont payables avec un Token ERC20 que vous aurez crée vous meme.
- Des opérateurs peuvent manuellement minter le nft (si le client souhaite payer par une autre méthode)
- Les opérateurs sont nommés par le owner du contrat. Ce privilège peut leur être retiré.
- Il peut y avoir plusieurs événements dans le contrat (on ne vas pas créer un contrat par évènement)
- La sécurité doit être la priorité. Même règles que pour le précédent.