Atelier Web 3.0 n°3: Apprendre à developper une cryptomonnaie et un NFT à l'aide d'un développeur Web 3.0

Sujet

Les bases

▼ Utilisez la bonne version

ℹ️ Il y a différentes versions de Solidity. Pour que le code compile bien, il faut préciser en haut du fichier:

```
pragma solidity ^0.8.0;
```

Ici ^0.8.0 autorise les verison supérieures à 0.8.0 de Solidity.

▼ Créer un contrat (+ constructor)

ℹ️ Un contrat est l'équivalent d'une classe en java ou en C et C++.
Il peut y en avoir plusieurs dans un seul fichier mais par soucis de visibilité, nous vous conseillons de faire un contrat par fichier.
Dans chaque contrat, il y a un unique constructeur.
Le code ci-dessous détaille comment créer un contrat et son constructeur.

```
pragma solidity ^0.8.0;

contract MyFirstContract {

        constructor() public{

        }
}
```

Le mot clé public représente le domaine d'utilisation de la variable ou du constructeur (scope en anglais), c'est à dire dans quelles parties du code on peut l'utiliser ou non.
Pour la déclaration de fonction, il existe différents mot clés: public, internal, external et private, nous ne détaillerons pas ces notions dans cet atelier, nous déclarons toutes les variables et fonctions en public.

▼ Créer une variable

ℹ️ Pour les variables comme pour les fonctions et noms de contrat, on utilise la norme CamelCase, c'est à dire qu'on utilise pas les tirets ou les underscores.
Pour déclarer une variable, on le déclare comme ci-dessus.
Par la même occasion, nous donnons une liste non exhaustive des types de variables:

```
pragma solidity ^0.8.0;

uint nomVariable = 3;

uint256 nomVariable = 3;

bool nomVariable = true;

string CeciEstUneString = 'Hello World';

uint [] public nomVariable = [1, 2, 3, 4, 5];
```

Point important, le mot clé pour les entiers est “uint” et non “int” car les entiers n'existe pas, seules les entiers positifs (unsigned int) existent.

”uint8” : entier non signé qui peut contenir une valeur maximale de 2^8-1. uint8 ne doit utiliser qu'un seul octet d'espace de stockage.

”uint256” entier non signé qui peut contenir une valeur maximale de 2^(256)-1. uint256 nécessite 32 octets d'espace de stockage.

Attention, les points virgules sont obligatoires en fin de ligne !

▼ Créer une fonction

ℹ️ Une fonction en solidity se crée de la manière suivante:

```
function Name(type parametre1, type parametre2,...)
public view {
        //code de la fonction
}
```

Le mot clé public a la même signification que pour les variables et View indique que l'on ne modifie pas l'état de la blockchain. Cela est le cas seulement pour les getters, lorsqu'une variable change on retire View.

Si la fonction retourne une valeur, on devra préciser le type de la valeur que la fonction retournera :

```
function Name(type parametre1, type parametre2,...)
public view returns(type){
        //code de la fonction
                return //value
}
```

Attention à bien utiliser “returns” dans la déclaration de la fonction et “return” dans le code de la fonction.

1️⃣ Utiliser la version 0.8.20 de Solidity

2️⃣ Créer un contrat WelcomeToSolidity, ajoutez y deux variables: une chaine de caractères “name” et un entier “age”.

3️⃣ Créer le constructeur du contrat qui associe nos deux variables précédentes à deux autres variables prises en paramètre du constructeur.
On rappelle que la convention pour les noms des arguments est de mettre un “_” en premier caractère.
De plus, il ne faut pas oublier de mettre le mot clé “memory” pour une chaîne de caractère exemple “string memory name”

4️⃣ Créer une fonction getMessage qui retourne la chaine de caractères suivante : la variable “name” concaténée avec la chaine de caractère “ fait ses premiers pas en solidity”.

5️⃣ Créer une fonction getResult qui crée deux entiers et retourne la somme des deux.

6️⃣ Créer une fonction sameParity ayant pour paramètre deux entiers. Si les deux entiers ont la même parité, la fonction retourne “true” sinon elle retourne “false”
Indications:
https://solidity-fr.readthedocs.io/fr/latest/types.html

✋ Faites vérifier votre code par Clément, Elie ou Noé

Nous allons maintenant rentrer dans le vif du sujet !
Nous vous laissons choisir dans un premier temps entre coder un NFT (Token ERC 721) et une cryptomonnaie (Token ERC 20).
Lorsque que vous avez fini, faites celui que vous n'avez pas choisi et si vous avez le temps, à la fin, se situe une partie bonus mêlant les deux notions.

Enjoy 😉

Développement d'un Token ERC 721

L'objectif ici est de créer un contrat associé à une collection NFT, ce contrat permettra d'obtenir un NFT, le transférer, gérer sa supply et de la connaitre à chaque instant.

1️⃣ Utilisez la version 0.8.20 de Solidity

2️⃣ Importer les librairies via l'url fourni
Nous vous encourageons à lire le contrat ERC 721 pour se familiariser avec les fonctions si vous le souhaitez.

```
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
```

3️⃣ Créez un contrat MyNFT qui hérite de ERC721.
Pour l'héritage, on utilise le mot clé “is”.

4️⃣ Créez trois variables globales: un entier non signé de taille 256 bits (”uint256”) appelé id, une variable de type “address” appelé owner

5️⃣ Le constructeur du contrat ERC721 prend cette forme:

```
constructor() ERC721("Nom du NFT", "Diminutif") {
        //instructions
    }
```

Dans ce constructeur, associez à la variable owner la personne qui appelle le contrat.

6️⃣ Créer une fonction mint qui permet à celui qui appelle le contrat de récupérer un NFT en utilisant la fonction suivante:

```
_safeMint(address, uint);
_safeMint("variable qui correspond à celui qui appelle le
```

De plus, on souhaite que l'on puisse mint au plus 10 NFT. On vous laisse cherch comment mettre cette condition 😊.

7️⃣ Créer une fonction “getNumberOfNFTMinted” qui retourne le nombre de NFT qui ont été mint

---

✋ Faites vérifier votre code par Clément, Elie ou Noé.

Développement d'un Token ERC 20

L'objectif de cette partie est de créer un jeton avec une supply de notre choix, de le transférer à une adresse, de mint le token et de changer le possesseur du contrat.

1️⃣ Utilisez la version 0.8.20 de Solidity

2️⃣ Importer les librairies via l'url fourni
Nous vous encourageons à lire le contrat ERC 20 pour se familiariser avec les fonctions si vous le souhaitez.

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

3️⃣ Créez un contrat MyToken qui hérite de ERC20.

4️⃣ Créez deux variables globales: un entier non signé de taille 256 bits appelé *_initial_*supply, une variable de type “address” public appelé owner. Affectez à l'entier non signé la supply que vous voulez.
Attention, on veut que notre token ait 18 décimales pour cela sous la forme uint : 1 token = 10 ** 18

5️⃣ Le constructeur du contrat ERC20 prend la même forme que celui de l'ERC 721.

Dans ce constructeur, associez à la variable owner la personne qui appelle le contrat et utilise la fonction du contrat ERC 20 qui permet de Mint, l'instruction doit permettre à la personne qui appelle le contrat de Mint toute la supply.

6️⃣ Créer une fonction Mint ayant pour argument une variable de type address et un unsigned int : uint. Cette fonction s'assure que l'owner est bien celui qui appelle le contrat et qui utilise la fonction Mint du contrat ERC 20 avec les arguments de la fonction.

7️⃣ Créer une fonction changeOwner qui a pour argument une address, celle-ci s'assure que l'owner est bien la personne qui appelle le contrat.
Cette fonction associé à la variable owner le paramètre de la fonction.

✋ Faites vérifier votre code par Clément, Elie ou Noé.

Bonus

❔ Rajouter une variable bool “trading”, lorsque cette variable est égale à “false”, seul l’owner peut appeler la fonction _transfer.
Ensuite, faites une fonction qui met la variable “trading” égale à true. Cette fonction ne peut être appelée que par l’owner.

❔ Rajouter 2 variables :

- maxWallet : nombre de tokens maximum qu’une personne peut posséder.
- maxAmount : montant de token maximum par transaction

Rajouter une variable “tax” qui correspond à un % de chaque transaction qui ira dans le wallet de l’owner.
Exemple : si tax = 5 alors 5% de chaque transaction ira dans le wallet de l’owner.

Modifier la fonction _transfer :

Aller dans le contrat ERC20.sol que l’on a importé, puis au niveau de la fonction “_transfer()” ajouter le mot clé “virtual” après le mot clé “internal”

Dans notre contrat, écrivez la fonction _transfer comme cela :

```
function _transfer(address from, address to,uint256 amount
        //modification que l'on veut apporter
        super._transfer(from, to, amount); //appel de la f
}
```

Le mot clé “override” permet de modifier la fonction _transfer du contrat ERC20.s

Lien entre token et NFT

❔ Déployer le contrat du NFT précédemment réalisé

❔ Créer un contrat Token et importer les contrats suivants, vous trouverez en bas de la page les contrats qu'il faut importer. Vous devez

- ERC20.sol
- ERC721.sol
- IUniswapV2Factory.sol
- IUniswapV2Router02.sol
- IERC20Metadata
- Context

❔ Récupérez l'adresse du contrat NFT que vous avez déployé et créez une variable "NFT" de type address qui sera égale à l'adresse du contrat NFT

❔ Modifier la fonction _beforeTokenTransfer(from, to, amount) du contrat ERC20 de sorte à ce que :

- si "from" ou "to" n'est pas le owner, il faut que "from" ou "to" possède le NFT.

❔ Ecrire une fonction doesUserHasNFT qui prend une adresse en paramètre et renvoie "true" si l'adresse possède le NFT et "false" sinon

Code vide :

```
pragma solidity ^0.8.0;

//import

contract Token is ERC20 {
    uint constant _initial_supply = 100000 * (10 ** 18);
```

```
    address public owner;
    address public sennder; // servira pour la fonction _beforeT
    uint256 public _amount; // servira pour la fonction _beforeT
    address public receiver; // servira pour la fonction _before
    address NFT = ; // adresse du contrat NFT

    mapping(address => bool) public exempt; // owner sera exempt

    address public lpPair; // adresse de la paire Token / WETH
    IUniswapV2Router02 public uniRouter;
    address _uniswap;

    constructor() ERC20("TokenTest", "TKN") {
        exempt[msg.sender] = true; // exclut l'owner, il n'aura
        _mint(msg.sender, _initial_supply); // mint la total sup
        owner = msg.sender;

        _uniswap = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D; /
        uniRouter = IUniswapV2Router02(_uniswap);
        lpPair = IUniswapV2Factory(uniRouter.factory()).createPa
            address(this),
            uniRouter.WETH()
        );
    }

    function _beforeTokenTransfer( // modification de la fonctio
        address from,
        address to,
        uint256 amount
    ) internal override {
        //...
          sennder = from;
          receiver = to;
          _amount = amount;

    }

    function mint(address to, uint256 amount) public {
        require(owner == msg.sender);
```

```
        _mint(to, amount);
    }

    function exemptAddress(address _address) external {
        exempt[_address] = true;
    }

    function changeOwner(address _owner) public {
        require(owner == msg.sender);
        owner = _owner;
    }

    function doesUserHasNFT(address _address) external view retu
        //...
    }
}
```

▼ **Contrats**

▼ IUniswapV2Factory

```
pragma solidity ^0.8.0;

interface IUniswapV2Factory {
    function createPair(
        address tokenA,
        address tokenB
    ) external returns (address pair);
}
```

▼ IUniswapV2Router02

```
pragma solidity ^0.8.0;

interface IUniswapV2Router02 {
    function factory() external pure returns (address);

    function WETH() external pure returns (address);
```

```
    function swapExactTokensForETHSupportingFeeOnTransferT
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;

    function swapExactETHForTokensSupportingFeeOnTransferT
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;

    function getAmountsOut(
        uint amountIn,
        address[] calldata path
    ) external view returns (uint[] memory amounts);

    function getAmountsIn(
        uint amountOut,
        address[] calldata path
    ) external view returns (uint[] memory amounts);
}
```

▼ ERC20

```
pragma solidity ^0.8.0;

import "interfaces/IERC20.sol";
import "interfaces/IERC20Metadata.sol";
import "contracts/Context.sol";

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are c
 * that a supply mechanism has to be added in a derived co
 * For a generic mechanism see {ERC20PresetMinterPauser}.
```

```
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.openzeppelin.com/t/how-to-implement-erc20
 * to implement supply mechanisms].
 *
 * The default value of {decimals} is 18. To change this,
 * this function so it returns a different value.
 *
 * We have followed general OpenZeppelin Contracts guideli
 * instead returning `false` on failure. This behavior is
 * conventional and does not conflict with the expectation
 * applications.
 *
 * Additionally, an {Approval} event is emitted on calls t
 * This allows applications to reconstruct the allowance f
 * by listening to said events. Other implementations of t
 * these events, as it isn't required by the specification
 *
 * Finally, the non-standard {decreaseAllowance} and {incr
 * functions have been added to mitigate the well-known is
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

    mapping(address => mapping(address => uint256)) privat

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * All two of these values are immutable: they can onl
     * construction.
     */
    constructor(string memory name_, string memory symbol_
        _name = name_;
```

```
        _symbol = symbol_;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual override returns (
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a sho
     * name.
     */
    function symbol() public view virtual override returns
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its
     * For example, if `decimals` equals `2`, a balance of
     * be displayed to a user as `5.05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the
     * Ether and Wei. This is the default value returned b
     * it's overridden.
     *
     * NOTE: This information is only used for _display_ p
     * no way affects any of the arithmetic of the contrac
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual override retur
        return 18;
    }

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view virtual override re
        return _totalSupply;
```

```
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view virtua
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     * - the caller must have a balance of at least `amoun
     */
    function transfer(address to, uint256 amount) public v
        address owner = _msgSender();
        _transfer(owner, to, amount);
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) pub
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * NOTE: If `amount` is the maximum `uint256`, the all
     * `transferFrom`. This is semantically equivalent to
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
```

```
    function approve(address spender, uint256 amount) publ
        address owner = _msgSender();
        _approve(owner, spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated al
     * required by the EIP. See the note at the beginning
     *
     * NOTE: Does not update the allowance if the current
     * is the maximum `uint256`.
     *
     * Requirements:
     *
     * - `from` and `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     * - the caller must have allowance for ``from``'s tok
     * `amount`.
     */
    function transferFrom(address from, address to, uint25
        address spender = _msgSender();
        _spendAllowance(from, spender, amount);
        _transfer(from, to, amount);
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to
     *
     * This is an alternative to {approve} that can be use
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated al
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
```

```
     */
    function increaseAllowance(address spender, uint256 ad
        address owner = _msgSender();
        _approve(owner, spender, allowance(owner, spender)
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to
     *
     * This is an alternative to {approve} that can be use
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated al
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of a
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 su
        address owner = _msgSender();
        uint256 currentAllowance = allowance(owner, spende
        require(currentAllowance >= subtractedValue, "ERC2
        unchecked {
            _approve(owner, spender, currentAllowance - su
        }

        return true;
    }

    /**
     * @dev Moves `amount` of tokens from `from` to `to`.
     *
     * This internal function is equivalent to {transfer},
     * e.g. implement automatic token fees, slashing mecha
     *
     * Emits a {Transfer} event.
     *
```

```
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     */
    function _transfer(address from, address to, uint256 a
        require(from != address(0), "ERC20: transfer from
        require(to != address(0), "ERC20: transfer to the

        _beforeTokenTransfer(from, to, amount);

        uint256 fromBalance = _balances[from];
        require(fromBalance >= amount, "ERC20: transfer am
        unchecked {
            _balances[from] = fromBalance - amount;
            // Overflow not possible: the sum of all balan
            // decrementing then incrementing.
            _balances[to] += amount;
        }

        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zer
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) intern
        require(account != address(0), "ERC20: mint to the

        _beforeTokenTransfer(address(0), account, amount);
```

```
        _totalSupply += amount;
        unchecked {
            // Overflow not possible: balance + amount is
            _balances[account] += amount;
        }
        emit Transfer(address(0), account, amount);

        _afterTokenTransfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reduc
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) intern
        require(account != address(0), "ERC20: burn from t

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amo
        unchecked {
            _balances[account] = accountBalance - amount;
            // Overflow not possible: amount <= accountBal
            _totalSupply -= amount;
        }

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }

    /**
```

```
     * @dev Sets `amount` as the allowance of `spender` ov
     *
     * This internal function is equivalent to `approve`,
     * e.g. set automatic allowances for certain subsystem
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint
        require(owner != address(0), "ERC20: approve from
        require(spender != address(0), "ERC20: approve to

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Updates `owner` s allowance for `spender` base
     *
     * Does not update the allowance amount in case of inf
     * Revert if not enough allowance is available.
     *
     * Might emit an {Approval} event.
     */
    function _spendAllowance(address owner, address spende
        uint256 currentAllowance = allowance(owner, spende
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= amount, "ERC20: in
            unchecked {
                _approve(owner, spender, currentAllowance
            }
        }
    }

    /**
     * @dev Hook that is called before any transfer of tok
```

```
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount`
     * will be transferred to `to`.
     * - when `from` is zero, `amount` tokens will be mint
     * - when `to` is zero, `amount` of ``from``'s tokens
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extend
     */
    function _beforeTokenTransfer(address from, address to

    /**
     * @dev Hook that is called after any transfer of toke
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount`
     * has been transferred to `to`.
     * - when `from` is zero, `amount` tokens have been mi
     * - when `to` is zero, `amount` of ``from``'s tokens
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extend
     */
    function _afterTokenTransfer(address from, address to,
}
```

▼ IERC20

```
pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the
 */
interface IERC20 {
    /**
```

```
     * @dev Emitted when `value` tokens are moved from one
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed t

    /**
     * @dev Emitted when the allowance of a `spender` for
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed

    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256)

    /**
     * @dev Returns the amount of tokens owned by `account
     */
    function balanceOf(address account) external view retu

    /**
     * @dev Moves `amount` tokens from the caller's accoun
     *
     * Returns a boolean value indicating whether the oper
     *
     * Emits a {Transfer} event.
     */
    function transfer(address to, uint256 amount) external

    /**
     * @dev Returns the remaining number of tokens that `s
     * allowed to spend on behalf of `owner` through {tran
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom}
     */
    function allowance(address owner, address spender) ext
```

```
    /**
     * @dev Sets `amount` as the allowance of `spender` ov
     *
     * Returns a boolean value indicating whether the oper
     *
     * IMPORTANT: Beware that changing an allowance with t
     * that someone may use both the old and the new allow
     * transaction ordering. One possible solution to miti
     * condition is to first reduce the spender's allowanc
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecom
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) exte

    /**
     * @dev Moves `amount` tokens from `from` to `to` usin
     * allowance mechanism. `amount` is then deducted from
     * allowance.
     *
     * Returns a boolean value indicating whether the oper
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address from, address to, uint25
}
```

▼ IERC20Metadata

```
pragma solidity ^0.8.0;

import "interfaces/IERC20.sol";

/**
 * @dev Interface for the optional metadata functions from
 *
 * _Available since v4.1._
 */
```

```
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view returns (string memory

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
}
```

▼ Context

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)

pragma solidity ^0.8.0;

/**
 * @dev Provides information about the current execution c
 * sender of the transaction and its data. While these are
 * via msg.sender and msg.data, they should not be accesse
 * manner, since when dealing with meta-transactions the a
 * paying for execution may not be the actual sender (as f
 * is concerned).
 *
 * This contract is only required for intermediate, librar
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (a
        return msg.sender;
    }

    function _msgData() internal view virtual returns (byt
```

```
            return msg.data;
        }
    }
```