Blockchain Developer

Table of Contents

15 An NFT project

Starting from the following repository on github, you can clone the project in your local environment. As usual can use Visual Code Studio to browse the code.

```
github clone https://github.com/PatrickAlphaC/nft-mix
```

You can play with the following commands:

```
brownie test
brownie compile
brownie run ./scripts/<deploy_something>
```

NFT were born as a standard to manage 'assets property'. They are 'non fungible' meaning they're all unique. You create a smart contract that manages 'token IDs', and for each of them you store its 'metadata' on the blockchain. You can't store a whole video or image on the blockchain, since it would be too expensive and it wouldn't scale. But you could for example store the video or the image hash, among other useful informations for you. They are compared to the concept of 'digital copyright', and despite of the hype and success they got in the short term period, in my opinion they are there to stay (not only for cryptokitties, that made NFT famous all around).

The picture below has been taken from the following site:
https://www.weforum.org/agenda/2022/02/non-fungible-tokens-nfts-and-copyright/
... that also details some interesting LEGAL information about copyright. It's not my area of interest, but there's a lot of material on the web.

Table 1

NFT Metadata

Item Metadata	
Contract Address	**Token Metadata**
0x8c5aCF6dBD24c66e6FD44d4A4C3d7a2 D955AA ad2	{ “symbol”: “Mintable Gasless store”, “image”: “https://d1czm3wxxz9zd.cloudfontnet/ 613b908d 0000000000/861932402826187638543675501608353605 31676033165 “animation_url”:””, “royalty_amount”:true, “address”: “0x8c5aCF6dBD24c66e6FD44d4A4C37a2D955AAad2”, “tokened” “861932402826187638543675016083536053 16760331 “resellable”: true, “original_creator”: “0xBe8Fa52a0A28AFE9507186A817813eDC1 “edition_number”:1, “description”: “ A beautiful bovine in the summer sun “auctionLength”: 43200, “title”: “The Clearest Light is the Most Blinding”, “url”: “https://metadata.mintable.app/mintable_gasless/86193 240 “file_key”:””, “apiURL”: “mintable_gasless/”, “name”: “The Clearest Light is the Most Blinding”, “auctionType”: “Auction”, “category”: “Art”, “edition_total”: 1, “gasless”: true }
Token ID	
8619324028261876385436755016083536053	
6083536053167603316518083457000846083267628374028898	
084608326762837402898	
Token Name	
The Clearest Light is the Most Blinding	
Original Image	
https://d1iczm3wxxz9zd.cloudfront.net/613b908d-19ad-41b1-8bfa0e0016820739c/0000000000000000/861932402826188763854367501608353605316760334571808345700084608326762837402898/ITEM_PREVIEW1.jpg	
Original Creator	
0xBe8Fa52a0A28AFE9507186A817813eDC14 54E004	

Image: Moringiello, Juliet M. and Odinet, Christopher K., *The Property Law of Tokens* (November 1, 2021). U Iowa Legal Studies Research Paper No. 2021-44 Used with permission.

Remember about NFT that:

- **storing data** on the Mainnet (Ethereum) is **expensive**, storing images, videos and other size consuming stuff is undoable from an economical perspective, and also from a technical one (the blockchain would explode ... which is why storing data on Ethereum is expensive, and there ‘L2 scaling mechanisms’ like ZK-rollups gaining success)
- to give you an idea about the previous point, practical implementations define often a ‘base_URI’ and a token_URI, the URI is being obtained by ‘base_URI + token_URI’ i.e. concatenating the two strings, to save SPACE on the mainnet.
- for this reason ‘**metadata’** is **stored** on the **blockchain**, only a few text description fields and an URI that points to the place where the ‘ASSET’ is stored
- it’s a good practice to store the real data on a **decentralized storing system** like for example IPFS (Inter Planetary File System), which of course is not the only one.

Not all Assets are managed in this way, this can lead to scams or bad practices: why should you buy something that is stored on the Personal Computer of the guy who sold it to you ?
Also remember that ‘Ownership’ is clear (the Ethereum account owning the NFT), authenticity of the pointed URI is not. Be careful to what you do and buy, don’t buy what you don’t know and understand.

15.1 Other details about NFT

Real life Dapps are usually specific implementations built with their own logic and extending the standard library, marketplaces are centralized and not "Distributed Apps" (probably it's the only way to manage it). The ERC721 standard doesn't define (and probably can't define) the application specific ways tokens are passed from an account to another. Some checks are done by openzeppeling libraries, but for example it can't be standardized that a specific NFT must have a price, or it can't impose an action and a time window to sell the Token (or better the **ASSET** that is associated to that specific token) or standardize a way to manage auctions.

https://docs.openzeppelin.com/contracts/2.x/erc721

On the official github page documentation:
https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/token/ERC721
we can find the following comment:
"This core set of contracts is designed to be unopinionated, allowing developers to access the internal functions in ERC721 (such as _mint) and expose them as external functions in the way they prefer. On the other hand, ERC721 Presets (such as {ERC721PresetMinterPauserAutoId}) are designed using opinionated patterns to provide developers with ready to use, deployable contracts."

From the above link, we could analyze the IERC721, where the 'I' stands for Interface. It defines the functions that need to be defined by the smart contract to be ERC721 compliant and compatible.

Some comments about the functions below:

- `balanceOf` is something different to the same function defined for an ERC20 token, which represents the number of tokens owned by the Ethereum account (usually they also have a value, and can be exchanged between accounts). This function returns the number of NFT for which a specific account is the owner, each of them being UNIQUE
- `ownerOf` returns the owner of a specific NFT identified by its unique (within the contract) token_ID
- the `SafeTransferFrom` function performs some checks before transfering the NFT, beware that the function IS NOT payable, thus doesn't manage the payment for the NFT, in case it's bought somewhere. This is INTENTIONALLY left to the specific implementations and needs. Also see the need for the transfer to be prior APPROVED by the owner of the token, in case this transaction is called by a third party (who also pays the Ethers for it)

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC721/IERC721.sol)

pragma solidity ^0.8.0;

import "../../utils/introspection/IERC165.sol";

/**
 * @dev Required interface of an ERC721 compliant contract.
 */
interface IERC721 is IERC165 {
    /**
     * @dev Returns the number of tokens in ``owner``'s account.
     */
    function balanceOf(address owner) external view returns (uint256 balance);

    /**
     * @dev Returns the owner of the `tokenId` token.
     *
     * Requirements:
```

```
     *
     * - `tokenId` must exist.
     */
    function ownerOf(uint256 tokenId) external view returns (address owner);

    /**
     * @dev Safely transfers `tokenId` token from `from` to `to`, checking first that contract recipients are aware of the ERC721 protocol to prevent tokens from being forever locked.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by `from`.
     * - If the caller is not `from`, it must be have been allowed to move this token by either {approve} or {setApprovalForAll}.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}, which is called upon a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId
    ) external;

... (output omitted)

    /**
     * @dev Gives permission to `to` to transfer `tokenId` token to another account.
     * The approval is cleared when the token is transferred.
     *
     * Only a single account can be approved at a time, so approving the zero address clears previous approvals.
     *
     * Requirements:
     *
     * - The caller must own the token or be an approved operator.
     * - `tokenId` must exist.
     *
     * Emits an {Approval} event.
     */
    function approve(address to, uint256 tokenId) external;


    /**
     * @dev Returns the account approved for `tokenId` token.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     */
    function getApproved(uint256 tokenId) external view returns (address operator);

...
```

… where there is the **function definition** to transfer the token ID's ownership from an address to another. Beware that we're talking about the token ID (an alphanumeric value that identifies the NFT), not tokens of other projects, exchanged on the market and with a money value. The proposed 'standard' **doesn't have payable functions**, this must be managed by those who define the contract. If you want to allow any possible token to pay, how would you do it ? everyone can create a new contract to create a new ERC20 token and become rich all of a sudden, but which is their value ? how would you know what is the right exchange rate ? would the owner agree on accepting tokens instead of Ethers ? it's all things you should manage in your smart contract and in your market, for example a seller could decide to set the price using a specific token

(but wouldn't accept others). 'Oracles' can be used to retrieve for example the right fiat/token ratio, given a starting price in Ethers ('Chainlink' was born for that, providing a decentralized Oracle systems).

You can also find the implementation here:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol
... again a few comments:

- an NFT must be **MINTED** (= created), in that moment it is associated to its creator and metadata is stored on the blockchain. Specific implementation could provide functions to change the reference URL of the NFT, or set a specific price to sell it using Ether or an ERC20 token
- the tokenURI function performs that string concatenation to save space on the blockchain
- an NFT can also be **BURNT** (=destroyed), always prior approval. In this case, the transfer is sent to address(0), that can't have any private key associated to it, thus to destroy tokens this is what is usually done.
- have a look to the storage variables that represent all necessary data and mappings

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.5.0) (token/ERC721/ERC721.sol)

pragma solidity ^0.8.0;

import "./IERC721.sol";
import "./IERC721Receiver.sol";
import "./extensions/IERC721Metadata.sol";
import "../../utils/Address.sol";
import "../../utils/Context.sol";
import "../../utils/Strings.sol";
import "../../utils/introspection/ERC165.sol";

/**
 * @dev Implementation of https://eips.ethereum.org/EIPS/eip-721[ERC721] Non-Fungible Token Standard, including
 * the Metadata extension, but not including the Enumerable extension, which is available separately as
 * {ERC721Enumerable}.
 */
contract ERC721 is Context, ERC165, IERC721, IERC721Metadata {
    using Address for address;
    using Strings for uint256;

    // Token name
    string private _name;

    // Token symbol
    string private _symbol;

    // Mapping from token ID to owner address
    mapping(uint256 => address) private _owners;

    // Mapping owner address to token count
    mapping(address => uint256) private _balances;

    // Mapping from token ID to approved address
    mapping(uint256 => address) private _tokenApprovals;

    // Mapping from owner to operator approvals
    mapping(address => mapping(address => bool)) private _operatorApprovals;

    /**
     * @dev Initializes the contract by setting a `name` and a `symbol` to the token collection.
```

```
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

...

    /**
     * @dev See {IERC721Metadata-tokenURI}.
     */
    function tokenURI(uint256 tokenId) public view virtual override returns (string
memory) {
        require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");

        string memory baseURI = _baseURI();
        return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI,
tokenId.toString())) : "";        <- baseURL concatenated to the string tokenID
    }

...

    /**
     * @dev Safely mints `tokenId` and transfers it to `to`.
     *
     * Requirements:
     *
     * - `tokenId` must not exist.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-
onERC721Received}, which is called upon a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function _safeMint(address to, uint256 tokenId) internal virtual {
        _safeMint(to, tokenId, "");
    }

    /**
     * @dev Same as {xref-ERC721-_safeMint-address-uint256-}[`_safeMint`], with an
additional `data` parameter which is
     * forwarded in {IERC721Receiver-onERC721Received} to contract recipients.
     */
    function _safeMint(
        address to,
        uint256 tokenId,
        bytes memory _data
    ) internal virtual {
        _mint(to, tokenId);
        require(
            _checkOnERC721Received(address(0), to, tokenId, _data),
            "ERC721: transfer to non ERC721Receiver implementer"
        );
    }

    /**
     * @dev Mints `tokenId` and transfers it to `to`.
     *
     * WARNING: Usage of this method is discouraged, use {_safeMint} whenever possible
     *
     * Requirements:
     *
     * - `tokenId` must not exist.
     * - `to` cannot be the zero address.
     *
     * Emits a {Transfer} event.
     */
    function _mint(address to, uint256 tokenId) internal virtual {
        require(to != address(0), "ERC721: mint to the zero address");
        require(!_exists(tokenId), "ERC721: token already minted");
```

```
        _beforeTokenTransfer(address(0), to, tokenId);

        _balances[to] += 1;
        _owners[tokenId] = to;

        emit Transfer(address(0), to, tokenId);

        _afterTokenTransfer(address(0), to, tokenId);
    }

    /**
     * @dev Destroys `tokenId`.
     * The approval is cleared when the token is burned.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     *
     * Emits a {Transfer} event.
     */
    function _burn(uint256 tokenId) internal virtual {
        address owner = ERC721.ownerOf(tokenId);

        _beforeTokenTransfer(owner, address(0), tokenId);

        // Clear approvals
        _approve(address(0), tokenId);

        _balances[owner] -= 1;
        delete _owners[tokenId];

        emit Transfer(owner, address(0), tokenId);

        _afterTokenTransfer(owner, address(0), tokenId);
    }


...
```

In the real world, NFT transfers should for example happen after the sender has received the payment in **Eth**, and once this payment has been performed, the transfer function must be invoked. This is the base of why smart contracts exist: ensure that something happens, if something else happens. There are many resources on the web, but you need to take care of what you read. They're not all expert, nor am I.

https://forum.openzeppelin.com/t/implementation-of-sellable-nft/5517

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/utils/Strings.sol";

contract NFTCollectible is ERC721 {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;
    address payable private _owner;
    uint256 private _maxSupply = 420;

    constructor(string memory myBase) ERC721("Collectible", "collect") {
        _setBaseURI(myBase);
        _owner = msg.sender;
    }
```

```
    function buyItem(address player) public payable returns (uint256)
    {
        require(totalSupply() < _maxSupply);
        require(msg.value == 0.2 ether, "Need to send exactly 0.2 ether");
        _tokenIds.increment();
        uint256 newItemId = _tokenIds.current();
        string memory newItemIdString = Strings.toString(newItemId);
        string memory metadata = "/metadata.json";
        string memory url = string(abi.encodePacked(newItemIdString, metadata));
        _safeMint(player, newItemId);
        _setTokenURI(newItemId, url);
        _owner.transfer(msg.value);        <- transfers 0.2Ether to the contract ?
        return newItemId;
    }

    function owner() public view returns (address) {
        return _owner;
    }

    function maxSupply() public view returns (uint256){
        return _maxSupply;
    }
}
```

15.2 Code comments

The following code represents a simple implementation of the NFT token ERC721. It uses everything already defined in the library, except for the function 'createCollectible' that creates an NFT referring to a specific URI. The function could be called by anyone the first time, the contract maintains an association between the tokenCounter index and the tokenURI. The standard library defines a few private variables to store the necessary informations, exposed if necessary through some functions.

```
    string private _name;

    // Token symbol
    string private _symbol;

    // Mapping from token ID to owner address
    mapping(uint256 => address) private _owners;

    // Mapping owner address to token count
    mapping(address => uint256) private _balances;

    // Mapping from token ID to approved address
    mapping(uint256 => address) private _tokenApprovals;

    // Mapping from owner to operator approvals
    mapping(address => mapping(address => bool)) private _operatorApprovals;
```

```
// SPDX-License-Identifier: MIT
pragma solidity 0.6.6;

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

contract SimpleCollectible is ERC721 {
    uint256 public tokenCounter;

    constructor() public ERC721("Dogie", "DOG") {
        tokenCounter = 0;
    }

    function createCollectible(string memory tokenURI) public returns (uint256)
    {
        uint256 newItemId = tokenCounter;
```

```
        _safeMint(msg.sender, newItemId);
        _setTokenURI(newItemId, tokenURI);
        tokenCounter = tokenCounter + 1;
        return newItemId;
    }
}
```

Further details can be found here:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol

How many other wrong parameters, addresses, variables could be passed to the above functions ? yeah ... this is why this world is so hard, and extensive testing is so important for it.

15.3 More on ERC721 standard

Since presently blockchain is like Internet in 1998, what is needed for sure are good standards for the applications. Openzeppeling is doing a great job, but I found out a wonderful article by "James Sangalli" talking about token's design.

https://medium.com/alphawallet/epic-fail-the-consequences-of-poor-erc-design-what-you-can-do-about-it-503e19c750

15.3.1 No ability to get token ids

If you look at erc721.org you will quickly notice that there is no functionality to get your token ids from the contract. You may not think this is a big deal but remember that the only way to make a transfer in erc721 is to reference your token id, without the ability to get your token ids, transferring is not even possible.
Further, the inability to get your token ids means you will have great difficulty displaying your tokens anywhere. The 'enumeration' interface exists in the standard, but is **optional**. This means nobody will implement it, thus leading to problems, mistakes, errors, and money lost in the worst case.

15.3.2 Inefficient transfer capability

The cryptokitties craze quickly showed the limit of the ethereum network and the fact that ERC721 only implements transferring tokens one by one exacerbated the problem to new heights.
This means that if you want to transfer 20 kitties to the same person, you need to send 20 transactions. This is very bad as in most cases, the majority of the gas cost comes from the transaction itself, meaning transferring tokens one by one will exponentially increase the cost and burden on the network.
This could have been mitigated if they had mapped balances to arrays and allowed these arrays to be transferred.

15.3.3 Inefficient design in general

To get your balance or token ids (if even implemented) requires looping through every single token in the contract and matching them back to your address. In the case of cryptokitties, this means indexing over 800k kitties just to get your own balance. This has caused node services like Infura to simply time out, leaving the user non the wiser on what their actual balance is.

15.3.4 What will happen if these problems are not addressed

But surely you can create extension functions to handle any shortcomings of a standard?
While it is true that you can extend the functionality to include whatever you want, you have to remember that people follow the standard they are given and whatever is missing will either be ignored or implemented in a subjective way. This means that major shortcomings like not having a way to get your token balance will

cause huge ripples going forward as such basic functionality will not be uniform (everyone's implementation will differ). Not addressing these issues will cause major issues for scalability (transfers and balance querying), adoption (lack of uniformity in implementation) and support (no way to be sure of how to perform basic functions properly). Going forward, we need to think hard about how we design our standards as poorly developed standards can become mainstream.

15.3.5 Solutions

So before you assume I am just a negative Nancy, let's talk about how we can fix this issue.
In general, it is best to only support a standard that handles basic functionality such as querying balance and transferring perfectly and not to worry about the bells and whistles as these can be extensions.
The fact that ERC721 has failed to handle basic functions means you should look for alternatives such as ERC875, ERC1155 or ERC998; all these standards enable efficient querying and transferring in a coherent manner and can be extended just like ERC721.
If we adopt other non fungible standards that get this right and choose good ERC's in general, we can fix problems like this.

16 A DAO project

This is a smart contract that uses tokens to provide the owners the chance to vote on something that is proposed by other people. Rules should be crystal clear in advance due to the smart contracts, voting should also be managed by such contracts so that results can't be tampered. Again the guys of OpenZeppeling have made a lot of work to provide libraries for DAO, that can be taken and re-used and extended for the specific applications. 'Curve' is a Defi project that is also a DAO, and provides such capabilities to possibly change details about the procotol (don't know much more about it right now).

https://www.youtube.com/watch?v=AhJtmUqhAqg

Starting from the following repository on github, you can clone the project in your local environment. As usual can use "Visual Code Studio" to browse the code.

```
github clone https://github.com/PatrickAlphaC/dao-template
```

From the readme file, we got the following steps:

1. We will deploy an ERC20 token that we will use to govern our DAO.
2. We will deploy a Timelock contract that we will use to give a buffer between executing proposals.
    - note: **The timelock is the contract that will handle all the money, ownerships, etc**
3. We will deploy our Governence contract
    - note: **The Governance contract is in charge of proposals and such, but the Timelock executes!**
4. We will deploy a simple Box contract, which will be owned by our governance process! (aka, our timelock contract).
5. We will propose a new value to be added to our Box contract.
6. We will then vote on that proposal.
7. We will then queue the proposal to be executed.
8. Then, we will execute it!

To accomplish the above simple governance protocol example, we have the following contracts:

- Box.sol: the contract storing the value, that is changed under proposal and voting
- GovernanceToken.sol: extends the ERC20votes of the library

- GovernanceTimeLock.sol: another library that controls the time and closes the possibility to vote after this pre-defined time is finished, for example 1 week
- GovernorContract.sol: this is the most important contract, since the other reuse what has been prepared by the guys of OpenZeppelin.

When you use libraries, it's a good practice to have a look to them and read the documentation, for example from the following:
https://docs.openzeppelin.com/contracts/4.x/api/governance

This is a base abstract contract that tracks voting units, which are a measure of voting power that can be transferred, and provides a system of vote delegation, where an account can delegate its voting units to a sort of "representative" that will pool delegated voting units from different accounts and can then use it to vote in decisions. In fact, voting units *must* be delegated in order to count as actual votes, and an account has to delegate those votes to itself if it wishes to participate in decisions and does not have a trusted representative. This contract is often combined with a token contract such that voting units correspond to token units. For an example, see ERC721Votes.

16.1 Solidity token contract

When you extend functions, sometimes you are forced to re-define and override them, but you can always simply perform the same. 'super' refers to the parent's implementation. Follows hereafter the implementation for the governance token, quite simple and straightforward.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";

contract GovernanceToken is ERC20Votes {
    uint256 public s_maxSupply = 1000_000_000_000_000_000_000_000_000;

    constructor()
        ERC20("GovernanceToken", "GT")
        ERC20Permit("GovernanceToken")
    {
        _mint(msg.sender, s_maxSupply);
    }

    // The functions below are overrides required by Solidity.

    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal override(ERC20Votes) {
        super._afterTokenTransfer(from, to, amount);
    }

    function _mint(address to, uint256 amount) internal override(ERC20Votes) {
        super._mint(to, amount);
    }

    function _burn(address account, uint256 amount) internal override(ERC20Votes)
    {
        super._burn(account, amount);
    }
}
```

16.2 Governor Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

import "@openzeppelin/contracts/governance/Governor.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorCountingSimple.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorVotes.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorVotesQuorumFraction.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorTimelockControl.sol";

// you can extend more than a single contract
contract GovernorContract is
    Governor,
    GovernorCountingSimple,
    GovernorVotes,
    GovernorVotesQuorumFraction,
    GovernorTimelockControl
{
    uint256 public s_votingDelay;
    uint256 public s_votingPeriod;

    constructor(
        ERC20Votes _token,
        TimelockController _timelock,
        uint256 _quorumPercentage,
        uint256 _votingPeriod,
        uint256 _votingDelay
    )
        Governor("GovernorContract")
        GovernorVotes(_token)
        GovernorVotesQuorumFraction(_quorumPercentage)
        GovernorTimelockControl(_timelock)
    {
        s_votingDelay = _votingDelay;
        s_votingPeriod = _votingPeriod;
    }

    function votingDelay() public view override returns (uint256) {
        return s_votingDelay; // 1 = 1 block
    }

    function votingPeriod() public view override returns (uint256) {
        return s_votingPeriod; // 45818 = 1 week
    }

    // The following functions are overrides required by Solidity.
    function quorum(uint256 blockNumber)
        public
        view
        override(IGovernor, GovernorVotesQuorumFraction)
        returns (uint256)
    {
        return super.quorum(blockNumber);
    }

    function getVotes(address account, uint256 blockNumber)
        public
        view
        override(IGovernor, GovernorVotes)
        returns (uint256)
    {
        return super.getVotes(account, blockNumber);
    }

    function state(uint256 proposalId)
        public
        view
        override(Governor, GovernorTimelockControl)
        returns (ProposalState)
    {
        return super.state(proposalId);
```

```
    }

    function propose(
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory calldatas,
        string memory description
    ) public override(Governor, IGovernor) returns (uint256) {
        return super.propose(targets, values, calldatas, description);
    }

    function _execute(
        uint256 proposalId,
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory calldatas,
        bytes32 descriptionHash
    ) internal override(Governor, GovernorTimelockControl) {
        super._execute(proposalId, targets, values, calldatas, descriptionHash);
    }

    function _cancel(
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory calldatas,
        bytes32 descriptionHash
    ) internal override(Governor, GovernorTimelockControl) returns (uint256) {
        return super._cancel(targets, values, calldatas, descriptionHash);
    }

    function _executor()
        internal
        view
        override(Governor, GovernorTimelockControl)
        returns (address)
    {
        return super._executor();
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        override(Governor, GovernorTimelockControl)
        returns (bool)
    {
        return super.supportsInterface(interfaceId);
    }
}
```

16.3 Deploy and run

```
from scripts.helpful_scripts import LOCAL_BLOCKCHAIN_ENVIRONMENTS, get_account
from brownie import (
    GovernorContract,
    GovernanceToken,
    GovernanceTimeLock,
    Box,
    Contract,
    config,
    network,
    accounts,
    chain,
)
from web3 import Web3, constants

# Governor Contract
QUORUM_PERCENTAGE = 4
# VOTING_PERIOD = 45818  # 1 week - more traditional.
```

```
# You might have different periods for different kinds of proposals
VOTING_PERIOD = 5  # 5 blocks
VOTING_DELAY = 1  # 1 block

# Timelock
# MIN_DELAY = 3600  # 1 hour - more traditional
MIN_DELAY = 1  # 1 seconds

# Proposal
PROPOSAL_DESCRIPTION = "Proposal #1: Store 1 in the Box!"
NEW_STORE_VALUE = 5


def deploy_governor():
    account = get_account()
    governance_token = (
        GovernanceToken.deploy(
            {"from": account},
            publish_source=config["networks"][network.show_active()].get(
                "verify", False
            ),
        )
        if len(GovernanceToken) <= 0
        else GovernanceToken[-1]
    )

    governance_token.delegate(account, {"from": account})
    print(f"Checkpoints: {governance_token.numCheckpoints(account)}")

    governance_time_lock = governance_time_lock = (
        GovernanceTimeLock.deploy(
            MIN_DELAY,
            [],
            [],
            {"from": account},
            publish_source=config["networks"][network.show_active()].get(
                "verify", False
            ),
        )
        if len(GovernanceTimeLock) <= 0
        else GovernanceTimeLock[-1]
    )

    governor = GovernorContract.deploy(
        governance_token.address,
        governance_time_lock.address,
        QUORUM_PERCENTAGE,
        VOTING_PERIOD,
        VOTING_DELAY,
        {"from": account},
        publish_source=config["networks"][network.show_active()].get("verify", False),
    )
    # Now, we set the roles...
    # Multicall would be great here ;)
    proposer_role = governance_time_lock.PROPOSER_ROLE()
    executor_role = governance_time_lock.EXECUTOR_ROLE()

    timelock_admin_role = governance_time_lock.TIMELOCK_ADMIN_ROLE()

    governance_time_lock.grantRole(proposer_role, governor, {"from": account})
    governance_time_lock.grantRole(
        executor_role, constants.ADDRESS_ZERO, {"from": account}
    )
    tx = governance_time_lock.revokeRole(
        timelock_admin_role, account, {"from": account}
    )
    tx.wait(1)
    # Guess what? Now you can't do anything!
    # governance_time_lock.grantRole(timelock_admin_role, account, {"from": account})
```

```
def deploy_box_to_be_governed():
    account = get_account()
    box = Box.deploy({"from": account})
    tx = box.transferOwnership(GovernanceTimeLock[-1], {"from": account})
    tx.wait(1)


def propose(store_value):
    account = get_account()
    # We are going to store the number 1
    # With more args, just add commas and the items
    # This is a tuple
    # If no arguments, use `eth_utils.to_bytes(hexstr="0x")`
    args = (store_value,)
    # We could do this next line with just the Box object
    # But this is to show it can be any function with any contract
    # With any arguments
    encoded_function = Contract.from_abi("Box", Box[-1], Box.abi).store.encode_input(
        *args
    )
    print(encoded_function)
    propose_tx = GovernorContract[-1].propose(
        [Box[-1].address],
        [0],
        [encoded_function],
        PROPOSAL_DESCRIPTION,
        {"from": account},
    )
    if network.show_active() in LOCAL_BLOCKCHAIN_ENVIRONMENTS:
        tx = account.transfer(accounts[0], "0 ether")
        tx.wait(1)
    propose_tx.wait(2)  # We wait 2 blocks to include the voting delay
    # This will return the proposal ID
    print(f"Proposal state {GovernorContract[-1].state(propose_tx.return_value)}")
    print(
        f"Proposal snapshot {GovernorContract[-
1].proposalSnapshot(propose_tx.return_value)}"
    )
    print(
        f"Proposal deadline {GovernorContract[-
1].proposalDeadline(propose_tx.return_value)}"
    )
    return propose_tx.return_value


# Can be done through a UI
def vote(proposal_id: int, vote: int):
    # 0 = Against, 1 = For, 2 = Abstain for this example
    # you can all the #COUNTING_MODE() function to see how to vote otherwise
    print(f"voting yes on {proposal_id}")
    account = get_account()
    tx = GovernorContract[-1].castVoteWithReason(
        proposal_id, vote, "Cuz I lika do da cha cha", {"from": account}
    )
    tx.wait(1)
    print(tx.events["VoteCast"])


def queue_and_execute(store_value):
    account = get_account()
    # time.sleep(VOTING_PERIOD + 1)
    # we need to explicity give it everything, including the description hash
    # it gets the proposal id like so:
    # uint256 proposalId = hashProposal(targets, values, calldatas, descriptionHash);
    # It's nearlly exactly the same as the `propose` function, but we hash the
description
    args = (store_value,)
    encoded_function = Contract.from_abi("Box", Box[-1], Box.abi).store.encode_input(
        *args
```

```
    )
    # this is the same as ethers.utils.id(description)
    description_hash = Web3.keccak(text=PROPOSAL_DESCRIPTION).hex()
    tx = GovernorContract[-1].queue(
        [Box[-1].address],
        [0],
        [encoded_function],
        description_hash,
        {"from": account},
    )
    tx.wait(1)
    tx = GovernorContract[-1].execute(
        [Box[-1].address],
        [0],
        [encoded_function],
        description_hash,
        {"from": account},
    )
    tx.wait(1)
    print(Box[-1].retrieve())


def move_blocks(amount):
    for block in range(amount):
        get_account().transfer(get_account(), "0 ether")
    print(chain.height)


def main():
    deploy_governor()
    deploy_box_to_be_governed()
    proposal_id = propose(NEW_STORE_VALUE)
    print(f"Proposal ID {proposal_id}")
    # We do this just to move the blocks along
    if network.show_active() in LOCAL_BLOCKCHAIN_ENVIRONMENTS:
        move_blocks(1)
    vote(proposal_id, 1)
    # Once the voting period is over,
    # if quorum was reached (enough voting power participated)
    # and the majority voted in favor, the proposal is
    # considered successful and can proceed to be executed.
    # To execute we must first `queue` it to pass the timelock
    if network.show_active() in LOCAL_BLOCKCHAIN_ENVIRONMENTS:
        move_blocks(VOTING_PERIOD)
    # States: {Pending, Active, Canceled, Defeated, Succeeded, Queued, Expired, Executed
}
    print(f" This proposal is currently {GovernorContract[-1].state(proposal_id)}")
    queue_and_execute(NEW_STORE_VALUE)
```

16.4 Output and debugging

```
PS C:\Users\<user>\PyScripts\DAO\dao_brownie> brownie run
.\scripts\governance_standard\deploy_and_run.py

INFORMAZIONI: impossibile trovare file corrispondenti ai
criteri di ricerca indicati.

Brownie v1.18.1 - Python development framework for Ethereum

DaoBrownieProject is the active project.

Launching 'ganache-cli.cmd --port 8545 --gasLimit 12000000 --accounts 10 --hardfork
istanbul --mnemonic brownie'...

Running 'scripts\governance_standard\deploy_and_run.py::main'...
```

```
...

Transaction sent: 0x884e2bfbb18fd856a83697d3eaa54a9122ec980133e82020452fd52fe954f5ae
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  Transaction confirmed   Block: 11   Gas used: 21000 (0.18%)

  Transaction confirmed   Block: 11   Gas used: 21000 (0.18%)

Required confirmations: 2/2
  GovernorContract.propose confirmed   Block: 10   Gas used: 81944 (0.68%)

Proposal state 1
Proposal snapshot 11
Proposal deadline 16
Proposal ID 89032801670644786090896159984413409335377199854927541874931809450777628720361
Transaction sent: 0xcae751379b114ce89c0fe217c7a6ba4bf853be8c8dcd2e1977117a5821d238f1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  Transaction confirmed   Block: 12   Gas used: 21000 (0.18%)

12
voting yes on
89032801670644786090896159984413409335377199854927541874931809450777628720361
Transaction sent: 0xd69df87e115cd4f511ebb1c9314c5e0d6adb402959ae83dedfc0c69ddce016b3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  GovernorContract.castVoteWithReason confirmed   Block: 13   Gas used: 79788 (0.66%)

OrderedDict([('voter', '0x66aB6D9362d4F35596279692F0251Db635165871'), ('proposalId',
89032801670644786090896159984413409335377199854927541874931809450777628720361),
('support', 1), ('weight', 100000000000000000000000), ('reason', 'Cuz I lika do da cha
cha')])
Transaction sent: 0x6d2388e7963ce4d2220d229bb7fb932ef4b56b68c26b39e585ee9deeebfcd9d2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  Transaction confirmed   Block: 14   Gas used: 21000 (0.18%)

...

18
 This proposal is currently 4
Transaction sent: 0x53bc5dc96442109c57da860bcb758214794f2aa4e76ed8126c34c72e83d7c830
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 18
  GovernorContract.queue confirmed   Block: 19   Gas used: 106643 (0.89%)
  GovernorContract.queue confirmed   Block: 19   Gas used: 106643 (0.89%)

Transaction sent: 0x49b541e8a410152b59653a2b535ad68c11740548f7d0da9cde64f56b2020b191
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 19
  GovernorContract.execute confirmed   Block: 20   Gas used: 109705 (0.91%)
  GovernorContract.execute confirmed   Block: 20   Gas used: 109705 (0.91%)

5
Terminating local RPC client...
```

We didn't get much about how things work from this output. To better understand what happens under the hood, we can launch 'brownie console' from the root project path, and execute all the commands contained in the mentioned python script, one by one (or copying more than that).

```
PS C:\Users\<user>\PyScripts\DAO\dao_brownie> brownie console
INFORMAZIONI: impossibile trovare file corrispondenti ai
criteri di ricerca indicati.
Brownie v1.18.1 - Python development framework for Ethereum

DaoBrownieProject is the active project.

Launching 'ganache-cli.cmd --port 8545 --gasLimit 12000000 --accounts 10 --hardfork
istanbul --mnemonic brownie'...
Brownie environment is ready.

>>> network.is_connected()
True
```

```
>>> accounts[0].address
'0x66aB6D9362d4F35596279692F0251Db635165871'

>>> accounts[0].balance()
100000000000000000000

>>> accounts[0].balance().to("ether")
Fixed('100.000000000000000000')
```

We can now copy ONE BY ONE all the lines of the script, also with the import statements, to check after every line what's going on.

```
>>> account = get_account()
>>> governance_token = (
        GovernanceToken.deploy(
            {"from": account},
            publish_source=config["networks"][network.show_active()].get(
                "verify", False
            ),
        )
        if len(GovernanceToken) <= 0
        else GovernanceToken[-1]
    )
Transaction sent: 0x77faedf64320d34a4d47ca369dc40dce5609da5db4d37b330edd3f95716e52f4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  GovernanceToken.constructor confirmed   Block: 1   Gas used: 1786814 (14.89%)
  GovernanceToken deployed at: 0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87

>>> governance_token.delegate(account, {"from": account})
Transaction sent: 0x6b44a478ed515b02e7ce19961dfee5305eb090c648e7f975c1a5d6eb039ddcf5
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  GovernanceToken.delegate confirmed   Block: 2   Gas used: 89944 (0.75%)

<Transaction '0x6b44a478ed515b02e7ce19961dfee5305eb090c648e7f975c1a5d6eb039ddcf5'>
>>>
```

We can go on assigning a proposer, an executor, a revoker. These are quite obvious roles. One other thing to modify respect to the previous code, is allowing more people (=accounts) to vote.

```
# Can be done through a UI
def vote(proposal_id: int, vote: int, account_idx: int):
    # 0 = Against, 1 = For, 2 = Abstain for this example
    # you can all the #COUNTING_MODE() function to see how to vote otherwise
    print(f"voting on {proposal_id} from account {account_idx}")
    account = get_account(index=account_idx)
    tx = GovernorContract[-1].castVoteWithReason(
        proposal_id, vote, "Cuz I lika do da cha cha", {"from": account})
    tx.wait(1)
    print(tx.events["VoteCast"])
```

16.5 Testing

Then in the run and deploy script, you can enjoy yourself playing for example in the following way:

```
def main():
    deploy_governor()
    deploy_box_to_be_governed()
    proposal_id = propose(NEW_STORE_VALUE)
    print(f"Proposal ID {proposal_id}")
    # We do this just to move the blocks along
    if network.show_active() in LOCAL_BLOCKCHAIN_ENVIRONMENTS:
        move_blocks(1)
    # we introduced here the possibility to vote for more accounts
    # 1 = for, 0=against, 2=don't care
    # below there are 2 votes for, 3 against, 2 don't care
    print("Starting to send votes")
```

```
    vote(proposal_id, 1, 0)
    vote(proposal_id, 0, 1)
    # this could be allowed, voting twice, the vote should override the previous one
    # if the proposal_id is the same of course
    vote(proposal_id, 2, 1)
    vote(proposal_id, 2, 2)
    vote(proposal_id, 0, 3)
    vote(proposal_id, 0, 7)
    vote(proposal_id, 0, 8)
    vote(proposal_id, 2, 9)
    # Once the voting period is over,
    # if quorum was reached (enough voting power participated)
    # and the majority voted in favor, the proposal is
    # considered successful and can proceed to be executed.
    # To execute we must first `queue` it to pass the timelock
    if network.show_active() in LOCAL_BLOCKCHAIN_ENVIRONMENTS:
        move_blocks(VOTING_PERIOD)
    # States: {Pending, Active, Canceled, Defeated, Succeeded, Queued, Expired, Executed
}
    print(
        f" This proposal is currently {GovernorContract[-1].state(proposal_id)}")
    queue_and_execute(NEW_STORE_VALUE)
```

If someone votes twice, the execution raises an error and stops:

```
OrderedDict([('voter', '0x33A4622B82D4c04a53e170c638B944ce27cffce3'), ('proposalId',
89032801670644786090896159984413409335377199854927541874931809450777628720361),
('support', 0), ('weight', 0), ('reason', 'Cuz I lika do da cha cha')])
voting on 89032801670644786090896159984413409335377199854927541874931809450777628720361
from account 1
Transaction sent: 0x0dbd30e08f128b60b080cf49d3d9deccd0a18277762943bc4d0cfb26f871a28d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  GovernorContract.castVoteWithReason confirmed (GovernorVotingSimple: vote already cast)
Block: 15   Gas used: 30968 (0.26%)
```

If all voters vote “2=don’t care”, when the proposal is queued for execution the transaction is reverted by the Governor smart contract.

```
This proposal is currently 3
Transaction sent: 0xf2344e074d51e2301cd3eb19fb2552b816b04ab2fcf2784fae5a34f88293c31f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 23
  GovernorContract.queue confirmed (Governor: proposal not successful)   Block: 30   Gas
used: 42483 (0.35%)
```

If 1 account votes for ‘yes’ and all the others “don’t care”, the result is the following:

```
Box before execution 0

Transaction sent: 0xc348badfa4ca6dcbac793168f0c0379d8b56c639ba0ed0d03fe0504477373056
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 24
  GovernorContract.execute confirmed   Block: 31   Gas used: 109705 (0.91%)

  GovernorContract.execute confirmed   Block: 31   Gas used: 109705 (0.91%)

Box after execution 5             <-- the proposed new value is changed !!!
```

Of course these are tests performed in the deploy python script, thus should be moved in a specific ‘test’ script in the tests directory.

16.6 Debugging

Moreover, the following voting sequence (VOTING_PERIOD has been raised to 10 to let more people vote, otherwise again you get an error, as it should be and this would be another test too):

```
vote(proposal_id, 1, 0)
```

```
vote(proposal_id, 0, 1)
vote(proposal_id, 0, 2)
vote(proposal_id, 0, 3)
vote(proposal_id, 2, 7)
vote(proposal_id, 2, 8)
vote(proposal_id, 2, 9)
```

... is successful. This apparently means that people voting 'against' or '0' are not properly handled. Once you enter such problems and you need to 'overload' libraries and debug them, you risk to enter a never ending loop. After a few more checks, it looks like **only the first account** can vote, while the others votes are discarded and not considered. Why ? debugging this can be a nightmare, you should start printing out everything in solidity, copying and pasting locally the imported smart contracts from openzeppelin.
One other thing to notice

```
tx = GovernorContract[-1].castVoteWithReason(proposal_id, vote, "Cuz I lika do da cha cha",
{"from": account})
```

You can find inside the "Governor.sol" the following function, which is not the one we are looging for since it's missing the account address of the voter ... **where is the right function ?** it's not in the GovernorContract.sol we have written (Patrick Collins did to be honest), so it must be found in any of the imported libraries, unless it can't in my opinion.

```
function castVoteWithReason(
        uint256 proposalId,
        uint8 support,
        string calldata reason     <-- we are missing the "from":account here !!!
    ) public virtual override returns (uint256) {
        address voter = _msgSender();
        return _castVote(proposalId, voter, support, reason);
    }


enum VoteType {
   Against,
   For,
   Abstain
}

    /**
     * @dev See {Governor-_countVote}. In this module, the support follows the `VoteType`
enum (from Governor Bravo).
     */
    function _countVote(
        uint256 proposalId,
        address account,
        uint8 support,
        uint256 weight,
        bytes memory // params
    ) internal virtual override {
        ProposalVote storage proposalvote = _proposalVotes[proposalId];

        require(!proposalvote.hasVoted[account],   "GovernorVotingSimple:   vote   already
cast");
        proposalvote.hasVoted[account] = true;

        if (support == uint8(VoteType.Against)) {
            proposalvote.againstVotes += weight;
        } else if (support == uint8(VoteType.For)) {
            proposalvote.forVotes += weight;
        } else if (support == uint8(VoteType.Abstain)) {
            proposalvote.abstainVotes += weight;
        } else {
            revert("GovernorVotingSimple: invalid value for enum VoteType");
        }
    }
```

Everything seems to be fine, but things do not work. Printing out the output of the transaction that creates the GovernanceToken, in particular after the call to the 'delegate' function, we see the following:

```
Events In This Transaction
--------------------------
└── GovernanceToken (0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87)
    ├── DelegateChanged
    │   ├── delegator: 0x66aB6D9362d4F35596279692F0251Db635165871
    │   ├── fromDelegate: 0x0000000000000000000000000000000000000000
    │   └── toDelegate: 0x66aB6D9362d4F35596279692F0251Db635165871
    └── DelegateVotesChanged
        ├── delegate: 0x66aB6D9362d4F35596279692F0251Db635165871    <- account[0] address
        ├── previousBalance: 0
        └── newBalance: 10000000000000000000000
```

After the voting transaction, this is the output:

```
---------------------
Tx Hash: 0xd69df87e115cd4f511ebb1c9314c5e0d6adb402959ae83dedfc0c69ddce016b3
From: 0x66aB6D9362d4F35596279692F0251Db635165871
To: 0x6951b5Bd815043E3F842c1b026b0Fa888Cc2DD85
Value: 0
Function: GovernorContract.castVoteWithReason
Block: 13
Gas Used: 79788 / 12000000 (0.7%)

Events In This Transaction
--------------------------
└── GovernorContract (0x6951b5Bd815043E3F842c1b026b0Fa888Cc2DD85)
    └── VoteCast
        ├── voter: 0x66aB6D9362d4F35596279692F0251Db635165871
        ├──proposalId:
89032801670644786090896159984413409335377199854927541874931809450777628720361
        ├── support: 1            <- vote 'for'
        ├── weight: 10000000000000000000000      <- voter's weight
        └── reason: Cuz I lika do da cha cha
```

If we vote with another account calling the 'vote' function, we discover the following:

```
Transaction was Mined
---------------------
Tx Hash: 0x1ce23269f247324775c66226b0fdf01ed92450a2e45d67687f26369000ccb0f5
From: 0x46C0a5326E643E4f71D3149d50B48216e174Ae84
To: 0x6951b5Bd815043E3F842c1b026b0Fa888Cc2DD85
Value: 0
Function: GovernorContract.castVoteWithReason
Block: 15
Gas Used: 56727 / 12000000 (0.5%)

Events In This Transaction
--------------------------
└── GovernorContract (0x6951b5Bd815043E3F842c1b026b0Fa888Cc2DD85)
    └── VoteCast
        ├── voter: 0x46C0a5326E643E4f71D3149d50B48216e174Ae84       <- no more account[0]
        ├──proposalId:
89032801670644786090896159984413409335377199854927541874931809450777628720361
        ├── support: 0
        ├── weight: 0    <-- LOOK HERE !!!!!!!!!!
        └── reason: Cuz I lika do da cha cha

None
OrderedDict([('voter',    '0x46C0a5326E643E4f71D3149d50B48216e174Ae84'),    ('proposalId',
89032801670644786090896159984413409335377199854927541874931809450777628720361),
('support', 0), ('weight', 0), ('reason', 'Cuz I lika do da cha cha')])
voting  on  89032801670644786090896159984413409335377199854927541874931809450777628720361
from account 6
```

Since the account 6 has not been delegated, probably its weight remains 0. Unfortunately, even if we call the function delegate right before making it vote, nothing changes. You can see that tokens are correctly sent from one account to the other, but no voting right is apparently allowed. The documentation in Openzeppelin site SEEMS to be rich and complete, but there is no example at all. The concept of delegation is not explained in a clear way, nor all the other concepts:
https://docs.openzeppelin.com/contracts/4.x/governance

"Once a proposal is active, delegates can cast their vote. Note that it is **delegates who carry voting power**: if a token holder wants to participate, they can set a trusted representative as their delegate, or they can become a delegate themselves by **self-delegating their voting power**."

The problem of this example, is that everything was performed by account[0]: this account created the GovernorToken and got the tokens. The functions mint, burn and tokenTransfer are ALL internal, and can't be called from the outside world. Moreover, to avoid problems and people cheating, the rights to vote depend on how many governor tokens you have on a specific block of the chain, in this case the creation's block. Thus we have more problems to solve here, to have more people capable of voting. The first problem was solved in the following way. I have launched the local Ganache IDE, and copied down the public address of accounts 1, 2 and 3.

```
contract GovernanceToken is ERC20Votes {
    uint256 public s_maxSupply = 1000_000_000_000_000_000_000_000;

    constructor()
        ERC20("GovernanceToken", "GT")
        ERC20Permit("GovernanceToken")
    {
        address temp;
        _mint(msg.sender, s_maxSupply);
        temp = address(0x94DF0324e5099410EeA66e1e0EA6C5a799D75275);
        _mint(temp, s_maxSupply);
        temp = address(0xFa4679DD96C885D5487363f1321420BE451c5299);
        _mint(temp, s_maxSupply);
        temp = address(0xDdB340364b1a012F972e59B54786858962801e88);
        _mint(temp, s_maxSupply);
    }
```

Then at the same time I didn't only provide tokens to msg.sender (which is account[0], who pays the gas to create the contract), but also to the other 3 accounts. Then as explained in the above documentation, I called the delegate function in the 'deploy_and_run.py':

```
account = get_account()
tx = governance_token = (
        GovernanceToken.deploy(
            {"from": account},
            publish_source=config["networks"][network.show_active()].get(
                "verify", False),
        )
        if len(GovernanceToken) <= 0
        else GovernanceToken[-1]
    )

    for ind in range(4):                    <-- debug purposes, check tokens
        account = get_account(ind)
        print(account.balance())

    for ind in range(4):
        tx = governance_token.delegate(
            get_account(ind),
           {"from": get_account(ind)}       <-- this is the transcation caller, msg.sender
```

```
        )
        print(tx.info())
```

Now with the following lines in the main function (the last parameters is the account's index):

```
print("Starting to send votes")
vote(proposal_id, 0, 0)           <-- 0 = vote against
vote(proposal_id, 1, 1)           <-- 1 = vote for
vote(proposal_id, 1, 2)           <-- 1 = vote for
vote(proposal_id, 2, 3)           <-- 2 = don't care
```

Follows the output:

```
Running 'scripts\governance_standard\deploy_and_run.py::main'...
Transaction sent: 0x772a3be3664d28c552c6d48ce8b5ccfb04bd699c90035740197d1eb851cc706a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 58
  GovernanceToken.constructor confirmed   Block: 92   Gas used: 1896326 (28.21%)
  GovernanceToken deployed at: 0x0566FB058f6457197a92e65f41c21c9A39B74c7D

10000000000000000000    <-- balance for GovernanceToken of account[0]
10000000000000000000    <-- balance for GovernanceToken of account[1]
10000000000000000000    <-- balance for GovernanceToken of account[2]
10000000000000000000    <-- balance for GovernanceToken of account[3]
```

Follows the logs of the 'delegate' transactions, they have tokens so they can delegate someone else to vote (tokens get transferred in this case), or they can delegate themselves:

```
Events In This Transaction
--------------------------
└── GovernanceToken (0x0566FB058f6457197a92e65f41c21c9A39B74c7D)
    ├── DelegateChanged
    │   ├── delegator: 0x165a38734a4453531eEA7c3483DdA4fC5852AaF1
    │   ├── fromDelegate: 0x0000000000000000000000000000000000000000
    │   └── toDelegate: 0x165a38734a4453531eEA7c3483DdA4fC5852AaF1
    └── DelegateVotesChanged
        ├── delegate: 0x165a38734a4453531eEA7c3483DdA4fC5852AaF1    <- account[0]
        ├── previousBalance: 0
        └── newBalance: 10000000000000000000000          <-- it gets 1M tokens

None
Checkpoints: 1
Transaction sent: 0xb71c7df8d7ad5d917d5d0fd413a3d9ce5616155d8c1cd53d9791a3480099f8d2
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 6
  GovernanceToken.delegate confirmed   Block: 94   Gas used: 89944 (1.34%)

Transaction was Mined
---------------------
Tx Hash: 0xb71c7df8d7ad5d917d5d0fd413a3d9ce5616155d8c1cd53d9791a3480099f8d2
From: 0x94DF0324e5099410EeA66e1e0EA6C5a799D75275
To: 0x0566FB058f6457197a92e65f41c21c9A39B74c7D
Value: 0
Function: GovernanceToken.delegate
Block: 94
Gas Used: 89944 / 6721975 (1.3%)

Events In This Transaction
--------------------------
└── GovernanceToken (0x0566FB058f6457197a92e65f41c21c9A39B74c7D)
    ├── DelegateChanged
    │   ├── delegator: 0x94DF0324e5099410EeA66e1e0EA6C5a799D75275
    │   ├── fromDelegate: 0x0000000000000000000000000000000000000000
    │   └── toDelegate: 0x94DF0324e5099410EeA66e1e0EA6C5a799D75275
    └── DelegateVotesChanged
        ├── delegate: 0x94DF0324e5099410EeA66e1e0EA6C5a799D75275
        ├── previousBalance: 0
        └── newBalance: 10000000000000000000000
```

```
None
Checkpoints: 1
Transaction sent: 0x5afa9ac3ee63903456f2db64589624a343c8324b4249a24c8ac997278fd8a3fd
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 6
  GovernanceToken.delegate confirmed   Block: 95   Gas used: 89944 (1.34%)

Transaction was Mined
---------------------
Tx Hash: 0x5afa9ac3ee63903456f2db64589624a343c8324b4249a24c8ac997278fd8a3fd
From: 0xFa4679DD96C885D5487363f1321420BE451c5299
To: 0x0566FB058f6457197a92e65f41c21c9A39B74c7D
Value: 0
Function: GovernanceToken.delegate
Block: 95
Gas Used: 89944 / 6721975 (1.3%)

Events In This Transaction
--------------------------
└── GovernanceToken (0x0566FB058f6457197a92e65f41c21c9A39B74c7D)
    ├── DelegateChanged
    │   ├── delegator: 0xFa4679DD96C885D5487363f1321420BE451c5299
    │   ├── fromDelegate: 0x0000000000000000000000000000000000000000
    │   └── toDelegate: 0xFa4679DD96C885D5487363f1321420BE451c5299
    └── DelegateVotesChanged
        ├── delegate: 0xFa4679DD96C885D5487363f1321420BE451c5299
        ├── previousBalance: 0
        └── newBalance: 10000000000000000000000000

None
Checkpoints: 1
Transaction sent: 0xe787ab5a69c2f56dff9aaeddc87683615600efacdaeefcb91f19560249993db8
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 21
  GovernanceToken.delegate confirmed   Block: 96   Gas used: 89944 (1.34%)

Transaction was Mined
---------------------
Tx Hash: 0xe787ab5a69c2f56dff9aaeddc87683615600efacdaeefcb91f19560249993db8
From: 0xDdB340364b1a012F972e59B54786858962801e88
To: 0x0566FB058f6457197a92e65f41c21c9A39B74c7D
Value: 0
Function: GovernanceToken.delegate
Block: 96
Gas Used: 89944 / 6721975 (1.3%)

Events In This Transaction
--------------------------
└── GovernanceToken (0x0566FB058f6457197a92e65f41c21c9A39B74c7D)
    ├── DelegateChanged
    │   ├── delegator: 0xDdB340364b1a012F972e59B54786858962801e88
    │   ├── fromDelegate: 0x0000000000000000000000000000000000000000
    │   └── toDelegate: 0xDdB340364b1a012F972e59B54786858962801e88
    └── DelegateVotesChanged
        ├── delegate: 0xDdB340364b1a012F972e59B54786858962801e88
        ├── previousBalance: 0
        └── newBalance: 10000000000000000000000000
```

Cool ... now Voting events are printed, together with the transactions info. We print here only the event logs, look at their public addresses, at their vote (the support variable), and the weight which reports the number of governorTokens hold by the voter:

```
voting on 36182874094400554940465629273993956229099756658419938235220127595234609504937
from account 0
OrderedDict([('voter', '0x165a38734a4453531eEA7c3483DdA4fC5852AaF1'), ('proposalId',
36182874094400554940465629273993956229099756658419938235220127595234609504937),
('support', 0), ('weight', 10000000000000000000000000), ('reason', 'Cuz I lika do da cha
cha')])
```

```
voting on 36182874094400554940465629273993956229099756658419938235220127595234609504937
from account 1
None
OrderedDict([('voter', '0x94DF0324e5099410EeA66e1e0EA6C5a799D75275'), ('proposalId',
36182874094400554940465629273993956229099756658419938235220127595234609504937),
('support', 1), ('weight', 100000000000000000000000), ('reason', 'Cuz I lika do da cha
cha')])
voting on 36182874094400554940465629273993956229099756658419938235220127595234609504937
from account 2

None
OrderedDict([('voter', '0xFa4679DD96C885D5487363f1321420BE451c5299'), ('proposalId',
36182874094400554940465629273993956229099756658419938235220127595234609504937),
('support', 1), ('weight', 100000000000000000000000), ('reason', 'Cuz I lika do da cha
cha')])

voting on 36182874094400554940465629273993956229099756658419938235220127595234609504937
from account 3

OrderedDict([('voter', '0xDdB340364b1a012F972e59B54786858962801e88'), ('proposalId',
36182874094400554940465629273993956229099756658419938235220127595234609504937),
('support', 2), ('weight', 100000000000000000000000), ('reason', 'Cuz I lika do da cha
cha')])
```

And this is the final outcome:

```
Box before execution 0
Transaction sent: 0x5235cef1950ef7d1133284be5f84a7731338d12b076e8ea9c5c0b3b6399e6def
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 77
  GovernorContract.execute confirmed   Block: 122   Gas used: 109693 (1.63%)

  GovernorContract.execute confirmed   Block: 122   Gas used: 109693 (1.63%)

Box after execution 5
```

Lesson learnt: read the documentation and understand all that you can about how the library was created, and things should work. Then test everything, and avoid wasting hours in debugging things because you're trying to use the library in the wrong way. Ideally, you should understand the code of the imported library.

17 A Defi staking project

Defi stands for Decentralized Finance, and means invest your money with pre-defined unbreakable rules, without trusting third parties that could steal your money, or promise future payments that they don't have. There's a lot of space for inventing new protocols and projects here, but just to provide an idea, I will copy here the explanation about Uniswap version 1 protocol, most of it has been taken by the Reddit user "Happiness maxi".

17.1 Uniswap version 1

In this case the idea behind the process is quite simple: if you want to swap token_a with token_b, the product of the amount of the two tokens must remain costant, so there is a mathematical rule that lies behind the process. Let's see all the details or jump at 17.2 for the simple staking app.

17.1.1 Order Books

Centralized exchanges like Binance or Coinbase or the NYSE use **order books** to facilitate transactions between buyers and sellers. Order books are essentially lists of limit buy orders and limit sell orders that clients have placed. A limit order is an open offer to buy or sell some amount of an asset at a price specified by the customer placing the limit order.

These limit order get consumed by market orders, which is the instant kind of order that happens when you just click "Buy" or "Sell". If you execute a market buy, you will immediately buy from whatever limit sell order in the book currently is offering the lowest price. If you execute a market sell, you will immediately sell to

whatever limit buy order in the book currently is offering the highest price. These market orders consume part or all of the limit order that was offering the best deal.
The exchange profits by charging fees in return for the service of matching these market orders with these sell orders.

The official price of the asset at any time is simply the cheapest limit sell order on the books at the time (ie: the current best bargain, or how cheaply you could buy some of this asset *right now*).

The most expensive limit buy order on the books is always lower-priced than the cheapest limit sell order, and the difference between them is known as the spread.

Price moves when market buy orders entirely consume the cheapest limit sell order. When this happens, the price becomes whatever price is offered by the *next* cheapest limit sell order.

For example, imagine if the cheapest limit sell order currently on the books (ie: the best bargain, which is what defines the current price) is a whale offering to sell 100 BTC at $40k.

Now imagine that another whale makes a market buy for 70 BTC. They would get all 70 at the price of 40k, and this actually wouldn't change the price by one penny, because 30 BTC are still being offered by the first whale at $40k, so $40k is still the best bargain, and thus is the price.

However, if instead the buyer bought 140 BTC, they would fully consume the seller's limit sell order, and the price would now become whatever the next-cheapest limit sell order is priced at (for example, $40,000.50), and the remaining 40 BTC that the buyer buys will thus be bought at a higher price than what the first 100 BTC were bought for (when the price changes in the middle of a transaction like this, it is called **price slippage**).

It is even possible that when the cheapest limit sell order is consumed, the next cheapest limit sell order is at a significantly higher price, **causing the price to instantly teleport a great distance**. This generally happens when there is low liquidity, which means a low density of limit orders on the books. This is why high **liquidity is important for a healthy market**.

This is an important concept to keep in mind: the price of a stock is NOT a weighted average of the price to which all the available stocks have been bought during time, but reflects **only the last price** to which stocks have been exchanged. The '**market capitalization**' is for sure an important factor when considering an investment, and it's the actual price of a stock multiplied for the total number of available stocks on the market. The crazy thing in my opinion, is to suppose that, especially if many stocks are available, the price of

ALL stocks is that one ... should everyone try to sell its stocks at that specific price, would they all get sold? If there is hypothetically a dead market moment, and a single guy sells (to himself, having two different accounts) 1/1.000.000 of a Bitcoin for 1.000.000$ per Bitcoin, is the price for a Bitcoin now 1 million? Apparently yes, that's exactly how it works. So an extremely low exchange market, can be subjects to such problems and will be MUCH more volatile.
This concludes how order books work on centralized exchanges. As you can imagine, since decentralization is a major theme in crypto, there has long been a desire to find a peer-to-peer alternative. This was finally made possible with the invention of the **automated market maker**, which led to the birth of DeFi. **AMMs** were inspired by the structure of traditional stock dealer markets like the Nasdaq (rather than broker markets like the NYSE).

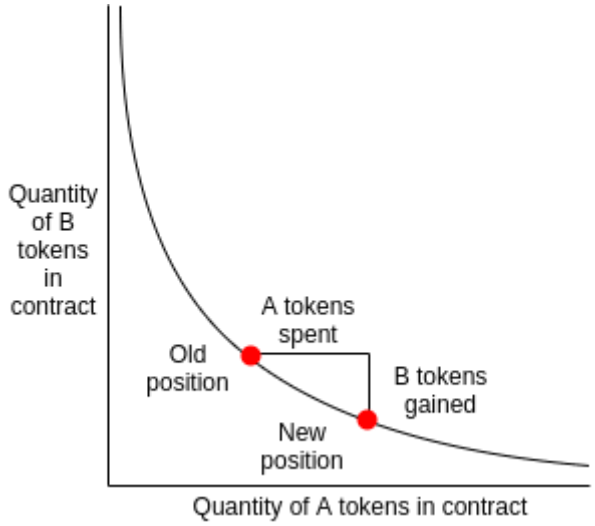
17.1.2 Automated Market Makers

AMMs are the innovation that lies at the core of every decentralized exchange, like UniSwap, SushiSwap, PancakeSwap, Curve and hundreds of others. AMMs use **smart contracts** to create an automatic, decentralized, peer-to-peer alternative to order books, allowing people to trade assets without going through CEXes.
The central idea of AMMs is a concept called liquidity pools. Each liquidity pool in an AMM allows people to trade a specific asset pair (like ETH/USDC) in either direction. In other words, an ETH/USDC liquidity pool would allow you to buy ETH with USDC or buy USDC with ETH. AMMs are made up of large amounts of these liquidity pools, allowing for large amounts of possible trade pairs.
Each liquidity pool is made up of equal portions of the trading pair's two assets. These pools are filled by liquidity providers, who are people like you and me who choose to supply their assets to facilitate trades by other people, in order to earn rewards in the form of trading fees.
When a trader uses the pool to make a swap, they are really just adding some amount to one of the two assets in the pool, and taking out the corresponding amount of the other asset in the pool. The trader also pays a trading fee, which is what rewards all the liquidity miners in that pool (they share the fee, weighted in proportion to how much of the pool each provider is providing).

As a side note, liquidity providers also sometimes get rewarded in a separate way if they provide liquidity to "incentivized pools". Sometimes, when some DEX or DeFi ecosystem is new, they will temporarily offer incentives to liquidity providers out of their own pocket in order to attract traders and gain a larger slice of the DeFi world, to profit more in the long run. These incentives usually follow a diminishing returns type of curve. Getting these rewards is called liquidity mining, and it is the central strategy in yield farming.

The description of liquidity pools I have provided so far is something a lot of you will have heard before. But it is missing a few key mechanics that I think are important to understand. If you are very sharp, then you might have thought of one or two questions when reading my explanation so far.
The two questions that I think we need to get to the bottom of before we truly understand liquidity pools are: what happens when the two halves of the pool are put out of balance due to traders using the pools to swap, and how does the pool know what relative price to use between the two assets ?
These are highly related questions. Here is the key: no matter what, the pool itself *always* considers the two sides of the pool (for example, the ETH side and the USDC side) to be of equal value. The following example is **EXTREMELY important** to better understand the concepts that usually lie behind AMM algorithms.
So, let's say you decide to buy ETH with USDC using a DEX. You want to spend $4000 USDC. The amount of ETH that will get you will depend on the ratio between the amount of ETH and the amount of USDC in the pool, and nothing else. Let's say the pool currently contains 1,000,000 USDC and 500 ETH. That is a ratio of 2000 USDC per 1 ETH. That means, in the pool's opinion, the price of ETH in USDC is 2000, regardless of what the outer world of CEXes and other DEXes might believe.

So, after your trade, you end up with 2 ETH, and the pool now contains 1,004,000 USDC and 498 ETH (plus a tiny bit extra, because your trading fee actually just gets added to the pool, and the providers will get their share of it whenever they pull their liquidity out).
Now the ratio of USDC to ETH in the pool is 2016, so the price of ETH in the pool's opinion is now 2016 USDC, and the price of USDC in the pool's opinion is 0.000496 ETH.
This brings us to a very key concept. The price of ETH in the pool's opinion has gone up to 2016 due to your trade, **but this price spike didn't happen in the rest of the world of CEXes and DEXes**! Therefore, the rest of the world probably still agrees that ETH costs about 2000 USDC, which brings an arbitrage opportunity: people can now buy discount USDC with their ETH from the pool in our example, and then use it to buy back their ETH plus a little extra on any other exchange. When people take advantage of this arbitrage opportunity, it pushes the price of ETH down (or equivalently the price of USDC up) in the eyes of the pool, reversing the effect of your trade, because they are adding ETH and removing USDC from the pool, bringing the ratio back towards 2000: 1.
The following two facts are extremely key:
The prices of the two assets in a pool are determined entirely **by the ratio between their amounts**. For example, if our pool somehow ended up containing 1 ETH and 1 million USDC (wouldn't happen because people would take advantage of arbitrage long before we could get there), then **the price of ETH in that pool would be 1 million USDC, regardless of the rest of the world**.
These arbitrage trades are the one and only thing that serve to rebalance the ratios of pools to keep the prices on DEXes more or less in lockstep with all other DEXes and CEXes. It basically makes it so that the average price in the eyes of the entire world acts as a point of gravity for any specific pool.



17.1.3 Being a liquidity provider

Generally speaking, anyone can create a new liquidity pool to allow others to trade some specific pair. Once a pool has been made, anybody can provide liquidity to it, or withdraw their liquidity, at any time. When you provide liquidity, you must provide the two assets in equivalent amounts (at least, in the eyes of the pool, determined by the current ratio of the pool).
When you provide liquidity, the funds leave your wallet, unlike with staking. This is necessary, because these funds need to be mobile to facilitate swaps.
So, how does the pool know that some portion of its liquidity belongs to you?
When you add liquidity to a pool, it will give you some amount of a special token called an LP (**Liquidity Pool**) token. The token will be specific to the asset pair, and will be called something like **LP-ETHUSDC**. They will also be **specific to the AMM** you are using.

These LP tokens are managed in such a way that the amount of this token that you, a liquidity provider, hold, is proportional to your slice of the pool. In other words, if you are providing 10% of all the liquidity in a pool, you will also have 10% of all LP-ETHUSDC tokens that exist on that AMM.
When you want to cash out, you trade in your LP tokens, and that lets the pool know how much ETH and USDC to give you back (in this example, you would get 10% of the ETH and 10% of the USDC in the pool, because you traded in 10% of all existing LP-ETHUSDC tokens, proving you owned 10% of the pool). Note that **trading fees are always just added to the pool**, making the total holdings of the pool go up, which means that when a liquidity provider pulls out their liquidity, the fees they earned while they were providing liquidity are naturally part of the share of the pool they have claim to.
Note that when you add your funds to a liquidity pool, you are taking on risk that the smart contract of the specific AMM you are using can be exploited. You are also exposed to a change in price of the two assets you are providing, because when you pull out your liquidity, it is given back to you in the form of those two assets. So it's like you were holding them all along.
So, in our example above, we are exposed to ETH price movements, we are exposed to USDC permanently losing its peg, and we are exposed to vulnerabilities in the smart contract of the DMM.
We are also always exposed to **one more key risk**.

17.1.4 Impermanent loss

Impermanent loss is a way that you can lose money when providing liquidity. More accurately, it refers to losing money *relative to if you had just held the two assets you provided to the pool*. In other words, you may gain money in an absolute sense due to the value of the assets in the pool going up, but because of impermanent loss, you might have gained more money by just holding.

In order for it to be worth it to provide liquidity, the trading fees you earn (plus any additional yield incentives you might be getting) must be enough to counteract the impermanent loss that will happen to you.

First I'll tell you when impermanent loss happens, and then I'll explain what it is.

Impermanent loss happens whenever the price of the two assets in the pool **change relative to each other**. The "relative to each other" part is really important. If the two assets go up in perfect lockstep together, or down together, or stay still together, then there is no impermanent loss. But if one goes up or down while the other doesn't move, or they go up or down together, but by different amounts, or (worst of all) one goes up while the other goes down, then you will experience impermanent loss.

Note that this means **providing liquidity for stable pairs like USDC/DAI** means you are basically **not exposed to impermanent loss** *or* price movements, assuming pegs hold. This is why those pools tend to offer far less reward.

Also note that stable/non-stable pairs are not necessarily more safe from impermanent loss that non-stable/non-stable pairs. With the latter, if the two assets tend to go up together and down together, then that pair will likely experience less impermanent loss than a stable/non-stable pair.

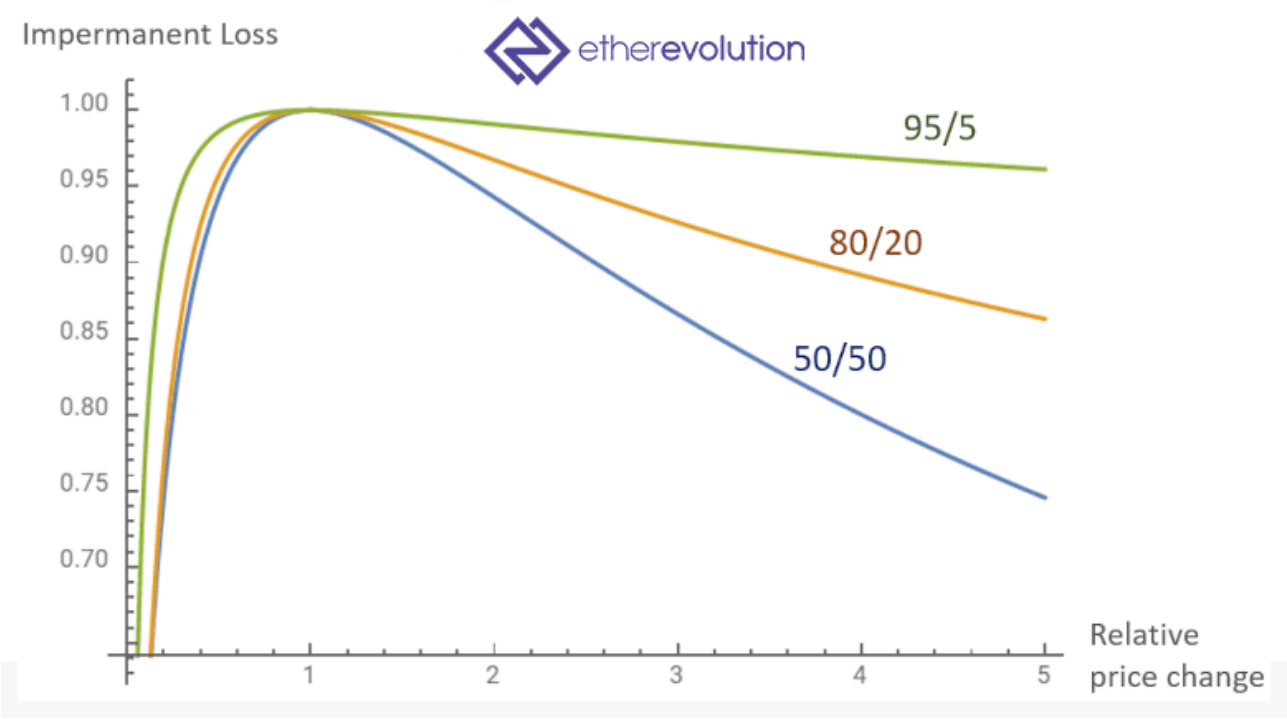
To understand what impermanent loss actually is, we need an example. Let's imagine two scenarios: one in which you just hold 1 ETH and 2000 USDC, and one in which you provide 1 ETH and 2000 USDC to a liquidity pool. Assume that the price of ETH is 2000 USDC at the time you provide to the pool, and that you own 10% of the pool. Thus, the pool must have 10 ETH and 20,000 USDC in it. Assume for simplicity that no other liquidity provider adds or removes liquidity to the pool while you are in it.

Now let's say the price of ETH in the eyes of the world spikes to 3000 USDC. This would cause arbitrage traders to quickly buy up 2 ETH from our pool for 2000 USDC each, because that would mean the pool now contains 8 ETH and, 24,000 USDC, which is a ratio of 3000 : 1. This means that our pool is now in agreement with the rest of the world, so we have found equilibrium, and there are no more arbitrage opportunities.

Now let's say you pull your liquidity. You own 10% of the pool, so you get 10% of the 8 ETH, and 10% of the 24,000 USDC. So, you get 0.8 ETH and 2400 USDC. Since ETH is worth 3000, the total value of your assets is (0.8 * 3000) + 2400 = $4800.

As for our holder: they still have 1 ETH and 2000 USDC, for a total of $5000.

So, **we lost $200** to impermanent loss by providing liquidity. Hopefully the trading fees and yield incentives were enough to offset that so that we are actually rewarded for taking more risk than holding.



17.2 A staking Dapp

Starting from the following repository on github, you can clone the project in your local environment. As usual can use “Visual Code Studio” to browse the code.

```
github clone https://github.com/PatrickAlphaC/defi-stake-yield-brown
```

As usual, you can play with the following commands:

```
brownie compile
brownie test
brownie run ./scripts/<deploy_something>
```

Follows an ‘in depth’ explanation and comments regarding the code. Smart contracts can be found under the ‘contracts’ directory. We want to create a smart contracts that ‘stakes’ balances for different accounts and users.

17.3 Dapp Token

```
pragma solidity ^0.8.0;

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

contract DappToken is ERC20 {
    constructor() public ERC20("Dapp Token", "DAPP"){
        _mint(msg.sender, 1000000000000000000000000);
    }
}
```

17.4 Token farm

Some comments regarding the code. All function should ideally be commented, this can generate automatically documentation on the code (see also openzeppling libraries). The contract is TokenFarm and is 'Ownable', this means it extends the library 'Ownable' that require for some special function that the creator of this contracts calls them, for security reasons. Comments have been added by me, not Patrick Collins who commented everything in its videos. The interesting part is that we use a 'Chainlink' oracle to read the price of a Token from the external world in a safe way, and get a realistic result. An Oracle is something that provides input data from the outside world (a blockchain doesn't know on its own how much dollars is an Ether).

```
/*
 starting from this solidity version there is no more need to use
 safeMath or similar, this uses slightly more gas to check for math
 operations that lead to overload, and revert the transaction
*/
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

contract TokenFarm is Ownable {
    // we use a pythonian dictionary of dictionaries where data is stored
    // in the following way:
    // mapping token's contract creator address --> staker address --> amount of tokens
    mapping(address => mapping(address => uint256)) public stakingBalance;

    // for every address we store the number of different Tokens staked,
    // for example someone could stake ETH, BTC, MATIC, ...
    mapping(address => uint48) public uniqueTokensStaked;

    // this is a dictionary, value for every token is an Oracle,
    // possibly e decentralized one with Chainlink or something else
    mapping(address => address) public tokenPriceFeedMapping;

    // public addresses of people that stored their tokens
    address[] public stakers;

    // lists all the available Tokens' contracts addresses that created them
    address[] public allowedTokens;

    // using the above openzeppeling library, create the IERC20 token
    IERC20 public dappToken;

    /*
     this is a generic 'dapp' roken built over ERC20 openzeppelin library
    */
    constructor(address _dappTokenAddress) public {
        dappToken = IERC20(_dappTokenAddress);
    }
```

```
    /*
    @dev this function sets the price given the token, but why should it be
    public ?!?! I guess for transparency reasons ...
    */
    function setPriceFeedContract(address _token, address _priceFeed)
        public
        onlyOwner
    {
        tokenPriceFeedMapping[_token] = _priceFeed;
    }

    /*
    this function is not really clear ...
    */
    function issueTokens() public onlyOwner {
        for (uint256 ind = 0; ind < stakers.length; ind++) {
            address recipient = stakers[ind];
            // we need to sum all the tokens of every staker, with the value
            // retrieved through on Oracle
            uint256 userTotValue = getUserTotalValue(recipient);
            // beware that the transfer function is inherited from openzeppelin
            dappToken.transfer(recipient, userTotValue);
        }
    }

    /*
    @dev for all tokens of a staker, retrieve the sum of the values of each one
    */
    function getUserTotalValue(address _user) public view returns (uint256) {
        uint256 totalValue = 0;
        require(uniqueTokensStaked[_user] > 0, "You're too poor bro' !!! ");
        // this function is slightly inefficient, since it asks for user's tokens
        // even in case the user has zero tokens of that specific type ...
        for (uint256 ind = 0; ind < allowedTokens.length; ind++) {
            totalValue += getUserSingleTokenValue(_user, allowedTokens[ind]);
        }
        return totalValue;
    }

    /*
    @dev
    */
    function getUserSingleTokenValue(address _user, address _token)
        public
        view
        returns (uint256)
    {
        if (uniqueTokensStaked[_user] <= 0) {
            return 0;
        }
        // this is an Oracle that asks how much (for example) Ether correpond to
        // how many Dapp tokens
        (uint256 price, uint256 decimals) = getTokenValue(_token);
        return ((stakingBalance[_token][_user] * price) / 10**decimals);
    }

    /*
    @dev here we need to recover from the outside world the real price information about
the
    token we are selling. Here Chainlink or any other 'Oracle' comes into play,
    through the below listed 'AggregatorV3Interface'
    */
    function getTokenValue(address _token)
        public
        view
        returns (uint256, uint8)
    {
        address priceFeedAddress = tokenPriceFeedMapping[_token];
        AggregatorV3Interface priceFeed = AggregatorV3Interface(
```

```
            priceFeedAddress
        );
        // this will cost some ChainLink tokens
        (, int256 price, , , ) = priceFeed.latestRoundData();
        return (uint256(price), priceFeed.decimals());
    }

    /*
    @dev two questions or things that could go wrong:
    - what tokens can be staked
    - how many tokens can be staked
    */
    function stakeTokens(uint256 _amount, address _token) public {
        // two questions or things that could go wrong:what tokens can be staked
        // how many tokens can be staked
        require(_amount > 0, "You can't stake nothing bro' !!!");
        require(tokenIsAllowed(_token), "Token is currently not allowed");

        // use the IERC20 interface from the library to transfer the money to be staken
        // from the function caller to this contract's address, for the requested amount
        IERC20(_token).transferFrom(msg.sender, address(this), _amount);
        updateUniqueTokensStaked(msg.sender, _token);
        // we update the number of tokens the account holds AFTER having updated the
        // tokens the account holds ... otherwise things do not work
        stakingBalance[_token][msg.sender] += _amount;
        // in case the staker is a new one, let's add it to the staker's array
        if (uniqueTokensStaked[msg.sender] == 1) {
            stakers.push(msg.sender);
        }
    }

    /**
    @dev this function is called by an external owner who has staked some tokens and
    wants to have them back.
    */
    function unstakeTokens(address _token) public {
        uint256 balance = stakingBalance[_token][msg.sender];
        require(balance > 0, "You're too poor bro' !!!");
        IERC20(_token).transfer(msg.sender, balance);
        stakingBalance[_token][msg.sender] = 0;
        // the following value should never be lower than 0, when it is 0
        // the address should be removed from the array too
        uniqueTokensStaked[msg.sender] -= 1;
        // here the address should be removed if uniqueTokensStaked == 0 ...
        if (uniqueTokensStaked[msg.sender] == 0) {
            for (uint256 ind = 0; ind < stakers.length; ind++) {
                if (stakers[ind] == msg.sender) {
                    stakers[ind] = stakers[stakers.length - 1];
                    stakers.pop();
                }
            }
        }
    }

    /**
    @dev only this contract can call this function, so it's internal. It can't be called
from another contract or account address. This function handles the fact that users can
stake many different tokens.
    */
    function updateUniqueTokensStaked(address _user, address _token) internal {
        if (stakingBalance[_token][_user] <= 0) {
            uniqueTokensStaked[_user] += 1;
        }
    }

    /*
    @dev
    */
    function addAllowedTokens(address _token) public onlyOwner {
        allowedTokens.push(_token);
```

```
    }

    /*
    @dev quite an expensive function ... should be understood if
    the allowedTokens could become a dictionary
    */
    function tokenIsAllowed(address _token) public returns (bool) {
        for (uint256 ind = 0; ind < allowedTokens.length; ind++) {
            if (allowedTokens[ind] == _token) {
                return true;
            }
        }
        return false;
    }
}
```

18 Tools

There are a lot of different tools that have to be used by blockchain developers, follows here a list. The problem is using the best ones, avoid loosing time with others. For example 'remix' can be used directly inside a browser, but it's not the best tool to manage things, since it stores things locally in browser. An IDE is available, but every time you make a change, you need to manually recompile all changed contracts, run test scripts, deploy the contract in the blockchain ... for this reason there are tools like Truffle, Hardhat and Brownie which perform all these kind of tasks with a single line command. This list has been copied from github and the Consensus team, just to provide an idea of how big this world already is.

18.1 New developers start here

- Solidity - The most popular smart contract language.
- Metamask - Browser extension wallet to interact with Dapps.
- **Truffle** - Most popular smart contract development, testing, and deployment framework. Install the cli via npm and start here to write your first smart contracts.
- Truffle boxes - Packaged components for the Ethereum ecosystem.
- **Hardhat** - Flexible, extensible and fast Ethereum development environment.
- Cryptotux - A Linux image ready to be imported in VirtualBox that includes the development tools mentionned above
- OpenZeppelin Starter Kits - An all-in-one starter box for developers to jumpstart their smart contract backed applications. Includes Truffle, OpenZeppelin SDK, the OpenZeppelin/contracts-ethereum-package EVM package of audited smart contract, a react-app and rimble for easy styling.
- EthHub.io - Comprehensive crowdsourced overview of Ethereum- its history, governance, future plans and development resources.
- EthereumDev.io - The definitive guide for getting started with Ethereum smart contract programming.
- **Brownie** - Brownie is a Python framework for deploying, testing and interacting with Ethereum smart contracts.
- Ethereum Stack Exchange - Post and search questions to help your development life cycle.
- dfuse - Slick blockchain APIs to build world-class applications.
- Biconomy - Do gasless transactions in your dapp by enabling meta-transactions using simple to use SDK.

- Blocknative — Blockchain events before they happen. Blocknative's portfolio of developers tools make it easy to build with mempool data.
- useWeb3.xyz — A curated overview of the best and latest resources on Ethereum, blockchain and Web3 development.

18.1.1 Developing Smart Contracts

18.1.1.1 Smart Contract Languages

- **Solidity** - Ethereum smart contracting language
- Vyper - New experimental pythonic programming language

18.1.1.2 Frameworks

- **Truffle** - Most popular smart contract development, testing, and deployment framework. The Truffle suite includes Truffle, Ganache, and Drizzle. Deep dive on Truffle here
- **Hardhat** - Flexible, extensible and fast Ethereum development environment.
- **Brownie** - Brownie is a Python framework for deploying, testing and interacting with Ethereum smart contracts.
- Embark - Framework for DApp development
- Waffle - Framework for advanced smart contract development and testing, small, flexible, fast (based on ethers.js)
- Dapp - Framework for DApp development, successor to DApple
- Etherlime - ethers.js based framework for Dapp deployment
- Parasol - Agile smart contract development environment with testing, INFURA deployment, automatic contract documentation and more. It features a flexible and unopinionated design with unlimited customizability
- 0xcert - JavaScript framework for building decentralized applications
- OpenZeppelin SDK - OpenZeppelin SDK: A suite of tools to help you develop, compile, upgrade, deploy and interact with smart contracts.
- sbt-ethereum - A tab-completey, text-based console for smart-contract interaction and development, including wallet and ABI management, ENS support, and advanced Scala integration.
- Cobra - A fast, flexible and simple development environment framework for Ethereum smart contract, testing and deployment on Ethereum virtual machine(EVM).
- Epirus - Java framework for building smart contracts.

18.1.1.3 IDEs

- **Remix** - Web IDE with built in static analysis, test blockchain VM.
- Ethereum Studio - Web IDE. Built in browser blockchain VM, Metamask integration (one click deployments to Testnet/Mainnet), transaction logger and live code your WebApp among many other features.
- Atom - Atom editor with Atom Solidity Linter, Etheratom, autocomplete-solidity, and language-solidity packages

- Vim solidity - Vim syntax file for solidity
- **Visual Studio Code** - Visual Studio Code extension that adds support for Solidity
- Ethcode - Visual Studio Code extension to compile, execute & debug Solidity & Vyper programs
- Intellij Solidity Plugin - Open-source plug-in for JetBrains IntelliJ Idea IDE (free/commercial) with syntax highlighting, formatting, code completion etc.
- YAKINDU Solidity Tools - Eclipse based IDE. Features context sensitive code completion and help, code navigation, syntax coloring, build in compiler, quick fixes and templates.
- Eth Fiddle - IDE developed by The Loom Network that allows you to write, compile and debug your smart contract. Easy to share and find code snippets.

18.1.2 Other tools

- Atra Blockchain Services - Atra provides web services to help you build, deploy, and maintain decentralized applications on the Ethereum blockchain.
- Azure Blockchain Dev Kit for Ethereum for VSCode - VSCode extension that allows for creating smart contracts and deploying them inside of Visual Studio Code

18.1.3 Test Blockchain Networks

- ethnode - Run an Ethereum node (Geth or Parity) for development, as easy as `npm i -g ethnode && ethnode`.
- **Ganache** - App for local test of Ethereum blockchain with visual UI and logs
- Kaleido - Use Kaleido for spinning up a consortium blockchain network. Great for PoCs and testing
- Besu Private Network - Run a private network of Besu nodes in a Docker container ** Orion - Component for performing private transactions by PegaSys ** Artemis - Java implementation of the Ethereum 2.0 Beacon Chain by PegaSys
- Cliquebait - Simplifies integration and accepting testing of smart contract applications with docker instances that closely resembles a real blockchain network
- Local Raiden - Run a local Raiden network in docker containers for demo and testing purposes
- Private networks deployment scripts - Out-of-the-box deployment scripts for private PoA networks
- Local Ethereum Network - Out-of-the-box deployment scripts for private PoW networks
- Ethereum on Azure - Deployment and governance of consortium Ethereum PoA networks
- Ethereum on Google Cloud - Build Ethereum network based on Proof of Work
- **Infura** - Ethereum API access to Ethereum networks (Mainnet, Ropsten, Rinkeby, Goerli, Kovan)
- CloudFlare Distributed Web Gateway - Provides access to the Ethereum network through the Cloudflare instead of running your own node
- Chainstack - Shared and dedicated Ethereum nodes as a service (Mainnet, Ropsten)
- Alchemy - Blockchain Developer Platform, Ethereum API, and Node Service (Mainnet, Ropsten, Rinkeby, Goerli, Kovan)

- ZMOK - JSON-RPC Ethereum API (Mainnet, Rinkeby, Front-running Mainnet)
- Watchdata - Provide simple and reliable API access to Ethereum blockchain

18.1.3.1 Test Ether Faucets

Blockchain to test contracts:

- Rinkeby faucet
- Kovan faucet
- Ropsten faucet (MetaMask)
- Ropsten faucet (rpanic)
- Goerli faucet
- Universal faucet
- Nethereum.Faucet - A C#/.NET faucet

18.1.4 Communicating with Ethereum

18.1.4.1 Frontend Ethereum APIs

- **Web3.js** - Javascript Web3
- Eth.js - Javascript Web3 alternative
- Ethers.js - Javascript Web3 alternative, useful utilities and wallet features
- useDApp - React based framework for rapid DApp development on Ethereum
- light.js A high-level reactive JS library optimized for light clients.
- Web3Wrapper - Typescript Web3 alternative
- Ethereumjs - A collection of utility functions for Ethereum like ethereumjs-util and ethereumjs-tx
- Alchemy-web3.js - Javascript Web3 wrapper with automatic retries, access to Alchemy's enhanced APIs, and robust websocket connections.
- flex-contract and flex-ether - Modern, zero-configuration, high-level libraries for interacting with smart contracts and making transactions.
- ez-ens - Simple, zero-configuration Ethereum Name Service address resolver.
- web3x - A TypeScript port of web3.js. Benefits includes tiny builds and full type safety, including when interacting with contracts.
- Nethereum - Cross-platform Ethereum development framework
- dfuse - A TypeScript library to use dfuse Ethereum API
- Drizzle - Redux library to connect a frontend to a blockchain
- Tasit SDK - A JavaScript SDK for making native mobile Ethereum dapps using React Native
- useMetamask - a custom React Hook to manage Metamask in Ethereum ĐApp projects
- WalletConnect - Open protocol for connecting Wallets to Dapps
- Subproviders - Several useful subproviders to use in conjunction with Web3-provider-engine (including a LedgerSubprovider for adding Ledger hardware wallet support to your dApp)
- ethvtx - ethereum-ready & framework-agnostic redux store configuration. docs

- Strictly Typed - Javascript alternatives
    - elm-ethereum
    - purescript-web3
- ChainAbstractionLayer - Communicate with different blockchains (including Ethereum) using a single interface.
- Delphereum - a Delphi interface to the Ethereum blockchain that allows for development of native dApps for Windows, macOS, iOS, and Android.
- Torus - Open-sourced SDK to build dapps with a seamless onboarding UX
- Fortmatic - A simple to use SDK to build web3 dApps without extensions or downloads.
- Portis - A non-custodial wallet with an SDK that enables easy interaction with DApps without installing anything.
- create-eth-app - Create Ethereum-powered front-end apps with one command.
- Scaffold-ETH - Beginner friendly forkable github for getting started building smart contracts.
- Notify.js - Deliver real-time notifications to your users. With built-in support for Speed-Ups and Cancels, Blocknative Notify.js helps users transact with confidence. Notify.js is easy to integrate and quick to customize.

18.1.4.2 Backend Ethereum APIs

- Web3.py - Python Web3
- Web3.php - PHP Web3
- Ethereum-php - PHP Web3
- Web3j - Java Web3
- Nethereum - .Net Web3
- Ethereum.rb - Ruby Web3
- rust-web3 - Rust Web3
- Web3.hs - Haskell Web3
- KEthereum - Kotlin Web3
- Eventeum - A bridge between Ethereum smart contract events and backend microservices, written in Java by Kauri
- Ethereumex - Elixir JSON-RPC client for the Ethereum blockchain
- Ethereum-jsonrpc-gateway - A gateway that allows you to run multiple Ethereum nodes for redundancy and load-balancing purposes. Can be ran as an alternative to (or on top of) Infura. Written in Golang.
- EthContract - A set of helper methods to help query ETH smart contracts in Elixir
- Ethereum Contract Service - A MESG Service to interact with any Ethereum contract based on its address and ABI.
- Ethereum Service - A MESG Service to interact with events from Ethereum and interact with it.
- Marmo - Python, JS, and Java SDK for simplifying interactions with Ethereum. Uses relayers to offload transaction costs to relayers.

- Ethereum Logging Framework - provides advanced logging capabilities for Ethereum applications and networks including a query language, query processor, and logging code generation
- Watchdata - Provide simple and reliable API access to Ethereum blockchain

18.1.4.3 Bootstrap/Out-of-Box tools

- Truffle boxes - Packaged components for the Ethereum ecosystem
- Create Eth App - Create Ethereum-powered frontend apps with one command
- Besu Private Network - Run a private network of Besu nodes in a Docker container
- Testchains - Pre-configured .NET devchains for fast response (PoA) ** Blazor/Blockchain Explorer - Wasm blockchain explorer (functional sample)
- Local Raiden - Run a local Raiden network in docker containers for demo and testing purposes
- Private networks deployment scripts - Out-of-the-box deployment scripts for private PoA networks
- Parity Demo-PoA Tutorial - Step-by-Step tutorial for building a PoA test chain with 2 nodes with Parity authority round consensus
- Local Ethereum Network - Out-of-the-box deployment scripts for private PoW networks
- Kaleido - Use Kaleido for spinning up a consortium blockchain network. Great for PoCs and testing
- Cheshire - A local sandbox implementation of the CryptoKitties API and smart contracts, available as a Truffle Box
- aragonCLI - aragonCLI is used to create and develop Aragon apps and organizations.
- ColonyJS - JavaScript client that provides an API for interacting with the Colony Network smart contracts.
- ArcJS - Library that facilitates javascript application access to the DAOstack Arc ethereum smart contracts.
- Arkane Connect - JavaScript client that provides an API for interacting with Arkane Network, a wallet provider for building user-friendly dapps.
- Onboard.js - Blocknative Onboard is the quick and easy way to add multi-wallet support to your project. With built-in modules for more than 20 unique hardware and software wallets, Onboard saves you time and headaches.
- web3-react - React framework for building single-page Ethereum dApps

18.1.4.4 Ethereum ABI (Application Binary Interface) tools

- Online ABI encoder - Free ABI encoder online service that allows you to encode your Solidity contract's functions and constructor arguments.
- ABI decoder - library for decoding data params and events from Ethereum transactions
- ABI-gen - Generate Typescript contract wrappers from contract ABI's.
- Ethereum ABI UI - Auto-generate UI form field definitions and associated validators from an Ethereum contract ABI
- headlong - type-safe Contract ABI and Recursive Length Prefix library in Java

- EasyDapper - Generate dapps from Truffle artifacts, deploy contracts on public/private networks, offers live customizable public page to interact with contracts.
- One Click dApp - Instantly create a dApp at a unique URL using the ABI.
- Truffle Pig - a development tool that provides a simple HTTP API to find and read from Truffle-generated contract files, for use during local development. Serves fresh contract ABIs over http.
- Ethereum Contract Service - A MESG Service to interact with any Ethereum contract based on its address and ABI.
- Nethereum-CodeGenerator - A web based generator which creates a Nethereum based C# Interface and Service based on Solidity Smart Contracts.
- EVMConnector - Create shareable contract dashboards and interact with arbitrary EVM-based blockchain functions, with or without an ABI.

18.1.4.5 Patterns & Best Practices

18.1.4.5.1 Patterns for Smart Contract Development

- Dappsys: Safe, simple, and flexible Ethereum contract building blocks
  - has solutions for common problems in Ethereum/Solidity, eg.
    - Whitelisting
    - Upgradable ERC20-Token
    - ERC20-Token-Vault
    - Authentication (RBAC)
    - ...several more...
  - provides building blocks for the MakerDAO or The TAO
  - should be consulted before creating own, untested, solutions
  - usage is described in Dapp-a-day 1-10 and Dapp-a-day 11-25
- OpenZeppelin Contracts: An open framework of reusable and secure smart contracts in the Solidity language.
  - Likely the most widely-used libraries and smart contracts
  - Similar to Dappsys, more integrated into Truffle framework
  - Blog about Best Practices with Security Audits
- Advanced Workshop with Assembly
- Simpler Ethereum Multisig - especially section *Benefits*
- CryptoFin Solidity Auditing Checklist - A checklist of common findings, and issues to watch out for when auditing a contract for a mainnet launch.
- aragonOS: A smart contract framework for building DAOs, Dapps and protocols
  - Upgradeability: Smart contracts can be upgraded to a newer version
  - Permission control: By using the `auth` and `authP` modifiers, you can protect functionality so only other apps or entities can access it
  - Forwarders: aragonOS apps can send their intent to perform an action to other apps, so that intent is forwarded if a set of requirements are met
- EIP-2535 Diamond Standard
  - Organize contracts so they share the same contract storage and Ethereum address.
  - Solves the 24KB max contract size limit.

    - Upgrade diamonds by adding/replacing/removing any number of functions in a single transaction.
    - Upgrades are transparent by recording them with a standard event.
    - Get information about a diamond with events and/or four standard functions.
- Clean Contracts - A guide to writing clean code

18.1.4.5.2 Upgradeability

- Blog von Elena Dimitrova, Dev at colony.io
    - https://blog.colony.io/writing-more-robust-smart-contracts-99ad0a11e948
    - https://blog.colony.io/writing-upgradeable-contracts-in-solidity-6743f0eecc88
- Aragon research blog
    - Library driven development
    - Advanced Solidity code deployment techniques
- OpenZeppelin on Proxy Libraries

18.1.5 Infrastructure

18.1.5.1 Ethereum Clients

- Besu - an open-source Ethereum client developed under the Apache 2.0 license and written in Java. The project is hosted by Hyperledger.
- Geth - Go client
- OpenEthereum - Rust client, formerly called Parity
- Aleth - C++ client
- Nethermind - .NET Core client
- Infura - A managed service providing Ethereum client standards-compliant APIs
- Trinity - Python client using py-evm
- Ethereumjs - JS client using ethereumjs-vm
- Seth - Seth is an Ethereum client tool—like a "MetaMask for the command line"
- Mustekala - Ethereum Light Client project of Metamask
- Exthereum - Elixir client
- EWF Parity - Energy Web Foundation client for the Tobalaba test network
- Quorum - A permissioned implementation of Ethereum supporting data privacy by JP Morgan
- Mana - Ethereum full node implementation written in Elixir.
- Chainstack - A managed service providing shared and dedicated Geth nodes
- QuickNode - Blockchain developer cloud with API access and node-as-a-service.
- Watchdata - Provide simple and reliable API access to Ethereum blockchain

18.1.5.2 Storage

- IPFS - Decentralised storage and file referencing
    - Mahuta - IPFS Storage service with added search capability, formerly IPFS-Store

    - OrbitDB - Decentralised database on top of IPFS
    - JS IPFS API - A client library for the IPFS HTTP API, implemented in JavaScript
    - TEMPORAL - Easy to use API into IPFS and other distributed/decentralised storage protocols
    - PINATA - The Easiest Way to Use IPFS
- Swarm - Distributed storage platform and content distribution service, a native base layer service of the Ethereum web3 stack
- Infura - A managed IPFS API Gateway and pinning service
- 3Box Storage - An api for user controlled, distrubuted storage. Built on top of IPFS and Orbitdb.
- Aleph.im - an offchain incentivized peer-to-peer cloud project (database, file storage, computing and DID) compatible with Ethereum and IPFS.

18.1.5.3 Messaging

- Whisper - Communication protocol for DApps to communicate with each other, a native base layer service of the Ethereum web3 stack
- DEVp2p Wire Protocol - Peer-to-peer communications between nodes running Ethereum/Whisper
- Pydevp2p - Python implementation of the RLPx network layer
- 3Box Threads - API to allow developers to implement IPFS persisted, or in memory peer to peer messaging.

18.1.6 Testing Tools

- Truffle Teams - Zero-Config continuous integration for truffle projects
- Solidity code coverage - Solidity code coverage tool
- Solidity coverage - Alternative code coverage for Solidity smart-contracts
- Solidity function profiler - Solidity contract function profiler
- Sol-profiler - Alternative and updated Solidity smart contract profiler
- Espresso - Speedy, parallelised, hot-reloading solidity test framework
- Eth tester - Tool suite for testing Ethereum applications
- Cliquebait - Simplifies integration and accepting testing of smart contract applications with docker instances that closely resembles a real blockchain network
- Hevm - The hevm project is an implementation of the Ethereum virtual machine (EVM) made specifically for unit testing and debugging smart contracts
- Ethereum graph debugger - Solidity graphical debugger
- Tenderly CLI - Speed up your development with human readable stack traces
- Solhint - Solidity linter that provides security, style guide and best practice rules for smart contract validation
- Ethlint - Linter to identify and fix style & security issues in Solidity, formerly Solium
- Decode - npm package which parses tx's submitted to a local testrpc node to make them more readable and easier to understand

- truffle-assertions - An npm package with additional assertions and utilities used in testing Solidity smart contracts with truffle. Most importantly, it adds the ability to assert whether specific events have (not) been emitted.
- Psol - Solidity lexical preprocessor with mustache.js-style syntax, macros, conditional compilation and automatic remote dependency inclusion.
- solpp - Solidity preprocessor and flattener with a comprehensive directive and expression language, high precision math, and many useful helper functions.
- Decode and Publish – Decode and publish raw ethereum tx. Similar to https://live.blockcypher.com/btc-testnet/decodetx/
- Doppelgänger - a library for mocking smart contract dependencies during unit testing.
- rocketh - A simple lib to test ethereum smart contract that allow to use whatever web3 lib and test runner you choose.
- pytest-cobra - PyTest plugin for testing smart contracts for Ethereum blockchain.

18.1.7 Security Tools

- MythX - Security verification platform and tools ecosystem for Ethereum developers
- Mythril - Open-source EVM bytecode security analysis tool
- Oyente - Alternative static smart contract security analysis
- Securify - Security scanner for Ethereum smart contracts
- SmartCheck - Static smart contract security analyzer
- Ethersplay - EVM disassembler
- Evmdis - Alternative EVM disassembler
- Hydra - Framework for cryptoeconomic contract security, decentralised security bounties
- Solgraph - Visualise Solidity control flow for smart contract security analysis
- Manticore - Symbolic execution tool on Smart Contracts and Binaries
- Slither - A Solidity static analysis framework
- Adelaide - The SECBIT static analysis extension to Solidity compiler
- solc-verify - A modular verifier for Solidity smart contracts
- Solidity security blog - Comprehensive list of known attack vectors and common anti-patterns
- Awesome Buggy ERC20 Tokens - A Collection of Vulnerabilities in ERC20 Smart Contracts With Tokens Affected
- Free Smart Contract Security Audit - Free smart contract security audits from Callisto Network
- Piet - A visual Solidity architecture analyzer

18.1.8 Monitoring

- Alethio - An advanced Ethereum analytics platform that provides live monitoring, insights and anomaly detection, token metrics, smart contract audits, graph visualization and blockchain search. Real-time market information and trading activities across Ethereum's decentralized exchanges can also be explored.
- amberdata.io - Provides live monitoring, insights and anomaly detection, token metrics, smart contract audits, graph visualization and blockchain search.

- Neufund - Smart Contract Watch - A tool to monitor a number of smart contracts and transactions
- Scout - A live data feed of the activities and event logs of your smart contracts on Ethereum
- Tenderly - A platform that gives users reliable smart contract monitoring and alerting in the form of a web dashboard without requiring users to host or maintain infrastructure
- Chainlyt - Explore smart contracts with decoded transaction data, see how the contract is used and search transactions with specific function calls
- BlockScout - A tool for inspecting and analyzing EVM based blockchains. The only full featured blockchain explorer for Ethereum networks.
- Terminal - A control panel for monitoring dapps. Terminal can be used to monitor your users, dapp, blockchain infrastructure, transactions and more.
- Ethereum-watcher - An extensible framework written in Golang for listening to on-chain events and doing something in response.
- Alchemy Notify - Notifications for mined and dropped transactions, gas price changes, and address activity for desired addresses.
- Blocknatve Mempool Explorer — Monitor any contract or wallet address and get streaming mempool events for every lifecycle stage — including drops, confirms, speedups, cancels, and more. Automatically decode confirmed internal transactions. And filter exactly how you want. Recieve events in our visual, no-code, interface or associate them with your API key to get events via a webhook. Mempool Explorer helps exchanges, protocols, wallets, and traders monitor and act on transactions in real-time.
- Ethernal - Ethereum block explorer for private chain. Browse transactions, decode function calls, event data or contract variables values on your locally running chain.

18.1.9 Other Miscellaneous Tools

- aragonPM - a decentralized package manager powered by aragonOS and Ethereum. aragonPM enables decentralized governance over package upgrades, removing centralized points of failure.
- Truffle boxes - Packaged components for building DApps fast.
    - Cheshire - A local sandbox implementation of the CryptoKitties API and smart contracts, available as a Truffle Box
- Solc - Solidity compiler
- Sol-compiler - Project-level Solidity compiler
- Solidity cli - Compile solidity-code faster, easier and more reliable
- Solidity flattener - Combine solidity project to flat file utility. Useful for visualizing imported contracts or for verifying your contract on Etherscan
- Sol-merger - Alternative, merges all imports into single file for solidity contracts
- RLP - Recursive Length Prefix Encoding in JavaScript
- eth-cli - A collection of CLI tools to help with ethereum learning and development
- Ethereal - Ethereal is a command line tool for managing common tasks in Ethereum
- Eth crypto - Cryptographic javascript-functions for Ethereum and tutorials to use them with web3js and solidity
- Parity Signer - mobile app allows signing transactions

- py-eth - Collection of Python tools for the Ethereum ecosystem
- truffle-flattener - Concats solidity files developed under Truffle with all of their dependencies
- Decode - npm package which parses tx's submitted to a local testrpc node to make them more readable and easier to understand
- TypeChain - Typescript bindings for Ethereum smartcontracts
- EthSum - A Simple Ethereum Address Checksum Tool
- PHP based Blockchain indexer - allows indexing blocks or listening to Events in PHP
- Purser - JavaScript universal wallet tool for Ethereum-based wallets. Supports software, hardware, and Metamask -- brings all wallets into a consistent and predictable interface for dApp development.
- Node-Metamask - Connect to MetaMask from node.js
- Solidity-docgen - Documentation generator for Solidity projects
- Ethereum ETL - Export Ethereum blockchain data to CSV or JSON files
- prettier-plugin-solidity - Prettier plugin for formatting Solidity code
- Unity3dSimpleSample - Ethereum and Unity integration demo
- Flappy - Ethereum and Unity integration demo/sample
- Wonka - Nethereum business rules engine demo/sample
- Resolver-Engine - A set of tools to standarize Solidity import and artifact resolution in frameworks.
- eth-reveal - A node and browser tool to inspect transactions - decoding where possible the method, event logs and any revert reasons using ABIs found online.
- Ethereum-tx-sender - A useful library written in Golang to reliably send a transaction — abstracting away some of the tricky low level details such as gas optimization, nonce calculations, synchronization, and retries.
- truffle-plugin-verify - Seamlessly verify contract source code on Etherscan from the Truffle command line.
- Blocknative Gas Platform — Gas estimation for builders, by builders. Gas Platform harnesses Blocknative's real-time mempool data infrastructure to accurately and consistently estimate Ethereum transaction fees. This provides builders and traders with an up-to-the-moment gas fee API.
- ETH Gas.watch - A gas price watcher with email notifications on price change

18.1.10 Smart Contract Standards & Libraries

18.1.10.1 ERCs - The Ethereum Request for Comment repository

- Tokens
  - ERC-20 - Original token contract for fungible assets
  - ERC-721 - Token standard for non-fungible assets
  - ERC-777 - An improved token standard for fungible assets
  - ERC-918 - Mineable Token Standard
- ERC-165 - Creates a standard method to publish and detect what interfaces a smart contract implements.

- ERC-725 - Proxy contract for key management and execution, to establish a Blockchain identity.
- ERC-173 - A standard interface for ownership of contracts

18.1.10.2 Popular Smart Contract Libraries

- Zeppelin - Contains tested reusable smart contracts like SafeMath and OpenZeppelin SDK library for smart contract upgradeability
- cryptofin-solidity - A collection of Solidity libraries for building secure and gas-efficient smart contracts on Ethereum.
- Modular Libraries - A group of packages built for use on blockchains utilising the Ethereum Virtual Machine
- DateTime Library - A gas-efficient Solidity date and time library
- Aragon - DAO protocol. Contains aragonOS smart contract framework with focus on upgradeability and governance
- ARC - an operating system for DAOs and the base layer of the DAO stack.
- 0x - DEX protocol
- Token Libraries with Proofs - Contains correctness proofs of token contracts wrt. given specifications and high-level properties
- Provable API - Provides contracts for using the Provable service, allowing for off-chain actions, data-fetching, and computation
- ABDK Libraries for Solidity - Fixed-point (64.64 bit) and IEEE-754 compliant quad precision (128 bit) floating-point math libraries for Solidity

18.1.11 Developer Guides for 2nd Layer Infrastructure

18.1.11.1 Scalability

18.1.11.2 Payment/State Channels

- Ethereum Payment Channel - Ethereum Payment Channel in 50 lines of code
- µRaiden Documentation - Guides and Samples for µRaiden Sender/Receiver Use Cases

18.1.11.3 Plasma

- Learn Plasma - Website as Node application that was started at the 2018 IC3-Ethereum Crypto Boot Camp at Cornell University, covering all Plasma variants (MVP/Cash/Debit)
- Plasma MVP - OmiseGO's research implementation of Minimal Viable Plasma
- Plasma MVP Golang - Golang implementation and extension of the Minimum Viable Plasma specification
- Plasma Guard - Automatically watch and challenge or exit from Omisego Plasma Network when needed.
- Plasma OmiseGo Watcher - Interact with Plasma OmiseGo network and notifies for any byzantine events.

18.1.11.4 Side-Chains

- POA Network
  - POA Bridge
  - POA Bridge UI
  - POA Bridge Contracts
- Loom Network
- Matic Network

18.1.11.5 Privacy / Confidentiality

18.1.11.5.1 ZK-SNARKs

- ZoKrates - A toolbox for zkSNARKS on Ethereum
- The AZTEC Protocol - Confidential transactions on the Ethereum network, implementation is live on the Ethereum main-net
- Nightfall - Make any ERC-20 / ERC-721 token private - open source tools & microservices
- Proxy Re-encryption (PRE) ** NuCypher Network - A proxy re-encryption network to empower data privacy in decentralized systems ** pyUmbral - Threshold proxy re-encryption cryptographic library
- Fully Homomorphic Encryption (FHE) ** NuFHE - GPU accelerated FHE library

18.1.11.6 Scalability + Privacy

18.1.11.7 ZK-STARKs

- StarkWare and StarkWare Resources - StarkEx scalability engine storing state transitions on-chain

18.1.11.8 Prebuilt UI Components

- aragonUI - A React library including Dapp components
- components.bounties.network - A React library including Dapp components
- ui.decentraland.org - A React library including Dapp components
- dapparatus - Reusable React Dapp components
- Metamask ui - Metamask React Components
- DappHybrid - A cross-platform hybrid hosting mechanism for web based decentralised applications
- Nethereum.UI.Desktop - Cross-platform desktop wallet sample
- eth-button - Minimalist donation button
- Rimble Design System - Adaptable components and design standards for decentralized applications.
- 3Box Plugins - Drop in react components for social functionality. Including comments, profiles and messaging.