# ERC20 - Smart Contract development













# Intro

- **Ethereum:** decentralized, open-source **blockchain** with **smart contract** functionality.
- **ERC20:** protocol for proposing the **common standard** for creating **fungible tokens** on the Ethereum blockchain.
- <u>Solidity</u>: Blockchain native / statically typed / supports inheritance / write operations cost gas

## Gas

- <u>Ethereum Gas</u> is a unit that measures the **amount of computational effort** that it will take to execute certain operations. The **lower** the Gas consumption the **better** is your smart contract.
- Gas price refers to the amount of ether you are willing to pay for every unit of gas.
- Gas limit refers to the maximum amount of gas you are willing to consume on a transaction.
- Ethereum 2.0 addresses some of the gas fee issues, which will in turn enable the platform to process thousands of transactions per second and scale globally.

## **Variables**

- Memory temporary place to store data and Gas consumption is not very significant.
- **Storage** these values get stored permanently on the blockchain and you can use across functions.
- Solidity declares variables in **storage** if keyword **memory** is not declared.
- Use **memory** for intermediate calculations and store the final result in **storage**.

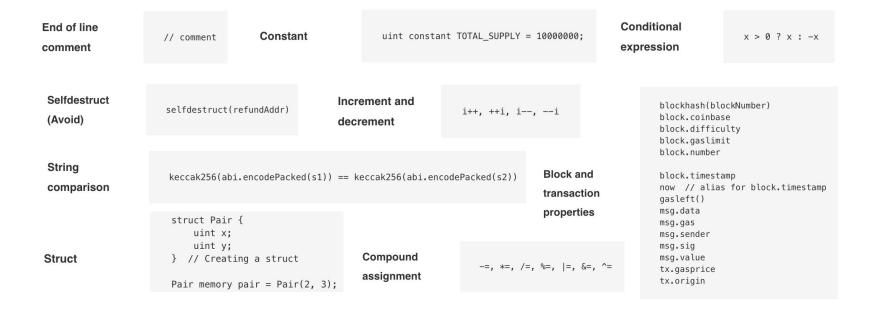
#### **Functions**

- **internal** can only be accessed internally (i.e. from within the current contract)
- external are part of the contract interface, which means they can be called from other contracts
   and via transactions
- **public functions** can be called internally from within the contract or externally via messages
- **private functions** are only visible for the contract they are defined in and not in derived contracts

# **Debugging / Events**

- Logging data from smart contracts with **Events**
- Events are dispatched signals which can be listened by anything connected to the network
- The logs generated by transactions are displayed in popular block explorers (eg. etherscan)
- Off chain scripts for listening to specific events and taking action when they occurs

#### **Cheat sheet**



## **Function Modifiers**

**Modifiers** can be used to change the behaviour of functions in a declarative way.

```
function createVoter(string paramName) onlyOwner public {
     Voter(paramName);
}
```

# Handling errors - Require/Revert usages

**Require** is convenient for checking inputs of a function especially in modifiers for example.

```
modifier onlyOwner() {
    require(msg.sender == owner. "Error msg");
    _;
}
// I'm checking if the caller of the function is really the owner of the smart contract
```

**Revert** is quite similar to require but does only accept one optional parameter: a string explaining why the

revert happened.

```
modifier onlyOwner() {
    if (msg.sender != owner) {
        revert("Error msg");
    }
    _;
}
```

# **Tooling**

- NodeJS / npm (Node v14.16.1)
- iterm2
- Visual Studio Code
- Solidity Visual Developer plugin
- <u>Ganache</u>
- <u>Truffle</u> (v5.3.7)
- **OpenZeppelin**
- **Etherscan**
- Web3js (v1.3.6)
- <u>Infura</u>
- [BONUS] Ethereum Developer Tools List by ConsenSys



## References

- https://www.bitpanda.com/academy/en/lessons/what-is-the-erc20-token-standard/
- https://blockgeeks.com/guides/ethereum-gas/
- <a href="https://ethereum.org/pt-br/developers/tutorials/logging-events-smart-contracts/">https://ethereum.org/pt-br/developers/tutorials/logging-events-smart-contracts/</a>
- <a href="https://reference.auditless.com/cheatsheet/solc-0-6-12-vyper-0-2-3/">https://reference.auditless.com/cheatsheet/solc-0-6-12-vyper-0-2-3/</a>