# Activity 2: Creating a Voting platform.

## Solidity basics

### Memory locations

There are three types of memory locations:

• **storage:** is used to permanently store data on the blockchain. State variables are stored in storage and have lifetime limited to the lifetime of the contract. Contract storage acts as a public database, from which values can be read with no fees. However, storing is expensive in terms of gas cost. Storage data is visible to all functions.

• **memory:** is used for temporary data required for local processing within functions and is less expensive in terms of gas cost. For some parameters, for example for string parameters it is mandatory to specify memory location.

• **calldata**: non-modifiable, non-persistent data; it is used for msg object and it’s the default location for parameters of external functions (functions that can only be called by other contracts).

### Solidity types

Solidity is statically typed, each variable has a type. Variables have default values, depending on their type.

**Value types** store variable value:

* Numeric types: integers **int/uint** and fixed-point number **fixed/unfixed**. Fixed/unfixed are not fully supported.
* Booleans: **bool** possible values true/false.
* Addresses **address** and **payable address.** Both address and payable address are 20 bytes vales of an Ethereum addresses and may be used to query balances, with member *balance*. Payable address has additional members: *transfer* and *send*. Payable addresses are addresses one can send eth to. To directly interact with contract code, address has members: *call*, *delegatecall* and *staticcall*.
* Fixed size byte arrays: **bytes1**, … **bytes32.** Fixed size byte arrays cost less than string in terms of gas.
* Enums are user defined types that assign names to an integral constants starting from 0.

**Reference types** store variable location:

* **mappings:** resembles hash tables. Each possible key exists, and it’s mapped to a default value (all bytes 0).
  + Syntax for mapping declaration:

mapping(KeyType KeyName? => ValueType ValueName?)

KeyName and ValueName are optional.KeyType can be any built-in value type, bytes, string, or any contract or enum type. ValueType can be any type.

* + keccak256 hash is used on key to look up the value mapped to the key.
  + Mappings can only have storage data location.
* **struct:** Custom defined types that group a set of variables.
* **arrays** can have a compile-time fixed size, or a dynamic size. We may append elements in a dynamically sized array calling member *push* and get the size of the array accessing member *length*. The index of the first element of an array is 0. Array elements can be of any type, including mapping or struct.
* **bytes** is usedfor arbitrary-length raw byte data
* **string** is used for arbitrary-length string (UTF-8) data**.**

### Contract structure and functions

Each contract contain: state variables declarations, struct types, enum types, functions, functions modifiers, events, and errors.

State variables cannot be modified by an external account, but their value is accessible for reading. Modifier public adds a default getter method.

**Access modifiers:**

**external**: function can be called only from another contract or using this. May not be specified for state variables.

**internal**: functions or state variables that may be accessed only within the contract and in derived contracts.

**private**: functions or state variables that may be accessed only within the contract.

**public**: functions or state variables that may be accessed by external user and by the contract. For public state variable a public getter is generated by default.

**Function modifiers**:

**view**: a view function is one that doesn't modify the state of the blockchain. A view function does not contain:

* write state variables statements.
* Emit event statements.
* Sending of eth.
* Creation of other contracts etc.

**pure**: functions that doesn't modify or read state variables. Pure functions usually include mathematical functions or formatting functions.

## Online voting.

An online voting system should allow a transparent and secure vote counting. The permission to vote should be granted to the rightful persons. Participants should not be manipulated during the voting period. The results should be accessible only when the voting period is over.

### EXERCISES

1. Work on file VotingStart.sol. Test on RemixIDE <https://remix.ethereum.org/>

* Add variable endVoting. Initialize this variable in the constructor using block.timestamp.
* Add variables or functions needed in order to verify that each generated voting token is used only once by a given address.
* Complete the definition of function winningProposal.

1. **Team members**: Modify the code to enable the registration and retrieval of team members names for each proposal.
2. **Delegated Votes**: Modify the code to enable each token owner to delegate it’s vote to another address after registration.

## Bibliography

<https://docs.soliditylang.org/_/downloads/en/latest/pdf/>

<https://web3-type-converter.onbrn.com/>