State Variables

Last Modified: November 04, 2025 - 2:35 PM

State variables are variables whose values are either permanently stored in contract storage or, alternatively, temporarily stored in transient storage which is cleaned at the end of each transaction.

Variable Location

The Ethereum Virtual Machine has different areas where it can store data: **storage**, **transient storage**, **memory**, and **the stack**.

Aspect	Memory	Storage	Transient Storage	Stack
Location	Temporary memory for functions.	Persistent blockchain storage.	Temporary storage (non-persistent).	EVM execution stack.
Lifetime	Function's execution.	Contract's lifetime.	Transaction's lifetime.	Operation's execution.
Persistence	Data is cleared after function.	Data persists across transactions.	Data is cleared after transaction.	Cleared after execution.
Gas Cost	Cheaper than storage.	Expensive.	Similar to storage, but temporary.	Very cheap.
Typical Use Case	Function arguments and local vars.	State variables.	Temporary mappings or buffers.	Intermediate calculations.

Variable Types (Documentation of Types)

Solidity is a statically typed language, which means that the type of each variable (state and local) needs to be specified.

The concept of “undefined” or “null” values does not exist in Solidity, but newly declared variables always have a default value dependent on its type.

1. State Variables of Value Types

These represent single values and are stored directly. They are **not references** to other objects.

1. **Numeric Types**:

- `uint`: Unsigned integer (`uint8`, `uint16`, … up to `uint256` in 8-bit increments).
- `int`: Signed integer (`int8`, `int16`, … up to `int256` in 8-bit increments).

2. **Fixed Point Types (Not Fully Supported)**:

- `fixed`: Signed fixed-point decimal (e.g., fixed128x18).

- `ufixed`: Unsigned fixed-point decimal (e.g., ufixed128x18).

3. **Address Types**:

- `address`: Represents an Ethereum address.
- `address payable`: A special address type that can send and receive Ether.

4. **Boolean**:

- `bool`: Represents `true` or `false`.

5. **Bytes**:

- `bytes1` to `bytes32`: Fixed-size byte arrays (from 1 to 32 bytes).
- `byte`: Alias for `bytes1`.
- `.length` yields the fixed length of the byte array (read-only).

6. **Enums**:

- Enums are one way to create a user-defined type in Solidity, for example:

```
enum Status { Pending, Shipped, Delivered }
```

2. State Variables of Reference Types

These represent complex data structures and are stored as references, pointing to memory or storage.

1. **Arrays**:

- Fixed-size arrays: `uint[5] fixedArray;`
- Dynamic-size arrays: `uint[] dynamicArray;`

2. **Mappings**:

- Key-value storage with a hash table structure. For example:

```
mapping(address => uint256) balances;
```

3. **Structs**:

- Custom types for grouping multiple variables. For example:

```
struct Person {
    string name;
```

```
    uint256 age;
}
```

4. **Strings**:

- A dynamic array of bytes used to store textual data: `string public name = "Alice";`

5. **Bytes (Dynamic)**:

- `bytes`: A dynamic array of bytes. `bytes public data;`

Variable Visibility

1. **public**: For variables you want to expose for external reading. Compiler automatically generates getter functions for them, which allows other contracts to read their values.
2. **internal**: Internal state variables can only be accessed from within the contract they are defined in and in derived contracts. They cannot be accessed externally. **This is the default visibility level for state variables.**
3. **private**: Private state variables are like internal ones but they are not visible in derived contracts.

Functions and Contracts

Last Modified: November 28, 2024 - 11:13 PM

Functions

Solidity supports defining function types. For example:

```
function add(uint256 num) public {

    result += num ;

}
```

Function Visibility

1. **public**: A function marked public can be called from anywhere (internal + external). **If no visibility specifier is given, functions are defaulted to public.**
2. **private**: A private function is accessible **only** within the contract where it is defined. It **cannot** be accessed from derived (child) contracts, nor from external callers.
3. **internal**: An internal function is accessible within the same contract and **from derived contracts** (via inheritance). It **cannot** be accessed by external accounts or contracts.
4. **external**: An external function can only be called by other contracts or external users. It **cannot** be called internally. The advantage of using `external` instead of `public` is to save gas fees.

Function Mutability

Define whether the function modifies the contract state or interacts with the blockchain:

1. **view**: Indicates the function does not modify the state but can read it. Run this function **does not** cost gas.
2. **pure**: Indicates the function does not modify or read the state. Run this function **does not** cost gas.
3. **payable**: Allows the function to receive Ether.

If we do not define a mutability specifier:

- The function is assumed to be able to modify the contract's state. Run this function **cost gas**.
- It is **not payable** unless explicitly marked with the payable keyword.

Function Components

A function contains the following components:

1. **Function Name**: the name of the function is used to call it.

2. **Parameters**:
   - Syntax: `type parameterName`
   - Multiple parameters are separated by commas.
3. **Visibility Specifiers**: define where the function can be called from: `public`, `private`, `internal`, `external`.
4. **Mutability Specifiers**: define whether the function modifies the contract state or interacts with the blockchain: `view`, `pure`, `payable`.
5. **Return Type**: specifies the type of data returned by the function. If multiple values are returned, they are specified as a tuple.
   - Syntax: `returns (type)`
   - Multiple return values: `returns (type1, type2, ...)`

```
function calculate(uint256 x, uint256 y) public returns (uint256) {

    return x + y;

}
```

Contract

A type used to represent a smart contract.

A contract can contains multiple functions. For example:

```
contract Add_Fun {

    uint256 result = 0;

    function add(uint256 num) public {

        result += num ;

    }
}
```