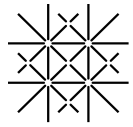

University of Basel

Center for Innovative Finance

Smart Contracts and Decentralized Finance

Solidity Basics

Prof. Dr. Fabian Schär
University of Basel

Release Ver.: (Local Release)
Version Hash: 3cb15c7281729de5ffcc7504865a11308da66536
Version Date: 2022-09-23 15:08:40 +0200

What is Solidity?

Solidity is . . .

- the most popular language for EVM Smart Contracts.
- influenced by C++, Python and JavaScript.
- statically typed, supports inheritance, libraries and complex user-defined types.
- still under rapid development with regular breaking changes.

Cutting edge tech

Use documentation frequently: https://docs.soliditylang.org

Solidity Version and the Pragma

Version of the Compiler

```
pragma solidity ^0.8.9;
```

- The pragma declares the version of the compiler to be used.
- The versioning follows Semantic Versioning 2.0.0: **Major.Minor.Patch**
- Breaking changes only come with new minor versions.
  → e.g., changes from 0.8.x to 0.9.0
- The ^ is commonly used and means "including any future version until the next minor release".
  → In our case: Any `ver` that satisfies `ver` $\geq$ 0.8.9 & `ver` $<$ 0.9.0
- Use the same pragma for all files in a project!

Basic Contract Structure

Basic Contract Structure

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.9;
3
4 contract HelloWorld {
5   // This is a comment
6   /*
7   This is a multiline comment
8   */
9 }
```

- **License identifier:** `UNLICENSED` is ok, but do not omit it.
- **Contract:** `contract <ContractName> {}`

Storing Data On-Chain: State Variables

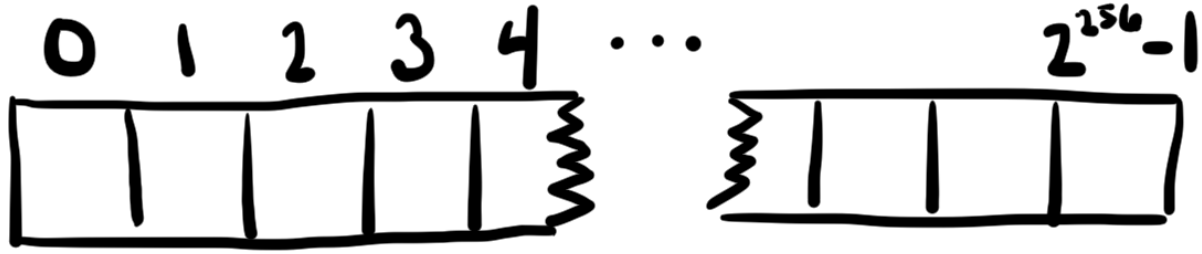

Figure: ↗ Understanding Ethereum Smart Contract Storage

Some types use multiple slots:

- Fixed size arrays and structs store their values in multiple subsequent slots, starting with their position.
- Dynamic size arrays store their length at their position and their values in multiple subsequent slots, starting with the hash of their position.
- Mappings store nothing at their position. The values are stored in the slots that correspond to the hash of the respective keys and the position of the mapping.

Integers

Integers

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.9;
3
4 contract HelloWorld {
5   uint public answer = 42;  // After deployment,
        this is stored on the blockchain
6 }
```

- `uint`, which is short for `uint256`, is the native integer.
- It is the most common type and can store a single number up to 32 bytes.
- Non-negative values from $[0, 2^{256} - 1]$ can be stored.

Integers

You can define both signed and unsigned integers:

- There exist smaller `uints`, like `uint8`, `uint16`, ..., `uint248`, but under normal circumstances they are all converted to `uint256` before use.
- `int` resp. `int256` can store negative numbers in half the range of `uint256`. Only use `int` when you explicitly need negative numbers.

More Elementary Types

A selection of more types:

- `address`: Stores a 20 byte Ethereum address.
- `bool`: Can be `true` or `false` and is used for logical operations.
- `bytes1`,. . . , `bytes32`: Stores arbitrary data of a fixed size.
- `fixed` / `ufixed`: Fixed point numbers are not fully supported yet. Do not use them!

We will look at more variable types as we use them.

Let's Build an Auction Platform

- One separate contract for each auction.
- Anyone should be able to participate.
- Start with a simple timed auction and explore other auction types later.

Auction Contract

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.9;
3
4 contract SimpleAuction {
5   // State of the auction
6   uint highestBid;
7   address highestBidder;
8   bool hasEnded;
9 }
```

Variable Default Values

Each variable has a default value when initialized.

Data Types	Zero Values
Integer types	0
bool	`false`
address	`0x0`
bytesX	`0x0`
array	[ ] (length = 0)
mapping	empty (no keys)
enum	(first choice)
string	""

E.g. `bool hasEnded;` is the same as `bool hasEnded = false;`

Exercise

Exercise 1

Use the `HelloWorld` contract from the `integer` section of this slide deck and familiarize yourself with the Remix IDE.

1. Create a new contract file.
2. Connect to the JavaScript VM, Goerli and your local Ganache blockchain.
3. Compile your contract.
4. Deploy your contract.
5. Read the `answer` `integer` value from the blockchain.