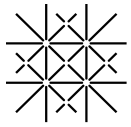

University of Basel

Center for Innovative Finance

Smart Contracts and Decentralized Finance

Functions and Modifiers

Prof. Dr. Fabian Schär
University of Basel

Release Ver.: (Local Release)
Version Hash: (None)
Version Date: (None)

Functions

Functions are executable parts of a smart contract.

Function Example

```
1   function <functionName>(<parameters>) <modifiers>
        returns (<return variables>) {
2     <function body>
3   }
```

`<functionName>`

- Pick and set a unique name for your function

`<parameters>`

- Used in the `<function body>`
- Must be different from state variables
- Recommendation: Use underscore as a prefix to differentiate between state variables and parameters

`<modifiers>`

- Accessibility modifiers
- State permission modifiers
- Special modifiers
- Custom modifiers

`<return variables>`

- Optional
- Multiple return variables possible

Accessibility Modifiers

Accessibility modifiers define which accounts can access a function or variable.

- They are explicitly required for all functions.
- They can be defined for variables, but default to `private` if omitted.

Who can access which functions/variables?

Keyword	EOA	This contract	Inheriting contract	External contract
`private`	No	Yes	No	No
`internal`	No	Yes	Yes	No
`external`	Yes	No	No	Yes
`public`	Yes	Yes	Yes	Yes

Additional information

- `external` can only be used for functions.
- Declaring a variable as `public` will create a getter function with the same name.
- `private` does NOT mean that the variable is hidden.

State Permission Modifiers

State permission modifiers define which functions can read from or modify (write to) the state.

Keyword	Read	Write
`pure`	No	No
`view`	Yes	No
<omitted>	Yes	Yes

Special and Custom Modifiers

Special modifiers for functions:

- `payable`: allows a function to receive ETH as part of the transaction.
- `virtual` & `override`: used for inheritance (more on inheritance later!).

Special modifiers for variables:

- `constant` & `immutable`: disallow changing the value of a variable during the contract's lifetime.
  - Difference: `immutable` can be set during contract deployment

Custom modifiers:

- Custom (user-defined) modifiers (more on this later!)

Update of the Auction Contract

Exercise 1:

1. Make sure the variables `beneficiary`, `highestBid`, `highestBidder` and `hasEnded` are `public`.
2. Create a function that allows any EOA or external contract to set the `beneficiary`.
3. Get the variable values.

Update the Auction Contract

Solution

```
1  contract SimpleAuction {
2    // Auction parameters
3    address public beneficiary;
4
5    // State of the auction
6    uint public highestBid;
7    address public highestBidder;
8    bool public hasEnded;
9
10   function initialize(address _beneficiary) external
         {
11     beneficiary = _beneficiary;
12   }
13 }
```

Contract Constructors

The `constructor` is a special function that is executed when the contract is deployed.

Constructor Example

```
1 constructor(<parameters>) {
2   <constructor body>
3 }
```

Immutable `beneficiary`

Exercise 2:

1. Remove `function initialize(address _beneficiary) external {...}` from the auction contract.
2. Use the `constructor` to set the `beneficiary` at the time of contract deployment for the contract's entire lifetime.

Solution:

Constructor for the Auction Contract

```
1 constructor(address _beneficiary) {
2   beneficiary = _beneficiary;
3 }
```

Line 3 in the previous contract

```
address public immutable beneficiary;
```

Current State of the Auction Contract

After Integrating the Constructor into `SimpleAuction`

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.9;
3
4  contract SimpleAuction {
5    // Auction parameters
6    address public immutable beneficiary;
7
8    // State of the auction
9    uint public highestBid;
10   address public highestBidder;
11   bool public hasEnded;
12
13   constructor (address _beneficiary) {
14     beneficiary = _beneficiary;
15   }
16 }
```