Solidity Cheat Sheet

by hsoudry via cheatography.com/36284/cs/18608/

Function Visibility

public	*accessible to all*
private	*accessible only to contract*
internal	*accessible to contract and subcontracts*
external	*accessible only outside contract*

Defining function visibility is mandatory.

Function Types

pure	*does not access the blockchain*
view	*does not modify the blockchain*
payable	*can receive Ether*

pure and *view* functions do not cost any gas.

Data Location

storage	*stored on the blockchain*
memory	*stored in memory*

Data location must be explicitely defined for all variables.
Storage is very expensive and must be used with caution.

Parameter Types

int / uint {8/256}

string

bool

address / address payable

Structures

```
struct StructureName {
    <parameter type> var1;
    <parameter type> var2;
    ...
}
```

Similar types should be grouped together in structures to lower gas cost.

Array and Mappings

```
<parameter type>[] arrayName;
mapping ( <parameter type> =>
<parameter type>) mappingName;
```

Arrays created in storage can have a variable size, arrays created in memory must have a fixed size at instantiation.
Array can be read through by indexes, mappings cannot.

Contract

```
contract contractName [is
inheritedContract,...] {...}
```

Constructor

```
constructor(<parameter types>)
{public|private|internal|external}
{...}
```

Constructors are optional, they are executed at contract creation.

Functions

```
function functionName(<parameter
types>)
{public|private|internal|external}
[pure|view|payable] [modifiers]
[returns (<return types>)] {...}
```

Interface

```
function functionName(<parameter
types>)
{public|private|internal|external}
[pure|view|payable] [modifiers]
[returns (<return types>)];
```

Definition must be identical to source function.

Modifiers

```
modifier modifierName(<parameter
types>) {...}
```

Use *_;* to continue with the function after running modifier code.

Events

```
event eventName(<parameter types>);
emit eventName(<parameters>);
```

Events are defined at contract root and emitted inside functions.

Useful links

- Remix IDE
- Solidity Documentation
- web3.js Documentation
- OpenZeppelin Contract Library

Security

Use Ownable contract to define owner of a contract and restrict usage of some functions using `onlyOwner` modifier.

Mind Overflow/Underflow when using integers. Use OpenZeppelin `SafeMath` library to prevent problems.

By **hsoudry**
cheatography.com/hsoudry/

Not published yet.
Last updated 29th January, 2019.