UNITS AND GLOBALLY AVAILABLE VARIABLES

ETHER UNITS

A literal number can take a suffix of wei, finney, szabo or ether to specify a subdenomination of Ether, where Ether numbers without a postfivx are assumed to be Wei.

```
assert(1 wei == 1);
assert(1 szabo == 1e12);
assert(1 finney == 1e15);
assert(1 ether == 1e18);
```

The only effect of the subdenomination suffix is a multiplication by a power of ten.

TIME UNITS

Suffixes like seconds, minutes, hours, days and weeks after literal numbers can be used to specify units of time where seconds are the base unit and units are considered naively in the following way:

```
1 == 1 seconds
1 minutes == 60 seconds
1 hours == 60 minutes
1 days == 24 hours
1 weeks == 7 days
```

Take care if you perform calendar calculations using these units, because not every year equals 365 days and not even every day has 24 hours because of leap seconds. Due to the fact that leap seconds cannot be predicted, an exact calendar library has to be updated by an external oracle.

These suffixes cannot be applied to variables. For example, if you want to interpret a function parameter in days, you can in the following way:

```
function f(uint start, uint daysAfter) public {
  if (now >= start + daysAfter * 1 days) {
    // ...
  }
}
```

BLOCK AND TRANSACTION PROPERTIES

- blockhash(uint blockNumber) returns (bytes32): hash of the given block - only works for 256 most recent, excluding current, blocks
- block.coinbase (address payable): current block miner's address
- block.difficulty (uint): current block difficulty
- block.gaslimit (uint): current block gaslimit
- block.number (uint): current block number
- block.timestamp (uint): current block timestamp as seconds since unix epoch
- gasleft() returns (uint256): remaining gas
- msg.data (bytes calldata): complete calldata
- msg.sender (address payable): sender of the message (current call)
- msg.sig (bytes4): first four bytes of the calldata (i.e. function identifier)
- msg.value (uint): number of wei sent with the message
- now (uint): current block timestamp (alias for block.timestamp)
- tx.gasprice (uint): gas price of the transaction
- tx.origin (address payable): sender of the transaction (full call chain)

ABI ENCODING AND DECODING FUNCTIONS

- abi.decode(bytes memory encodedData, (...)) returns (...): ABI-decodes the given data, while the types are given in parentheses as second argument. Example: (uint a, uint[2] memory b, bytes memory c) = abi.decode(data, (uint, uint[2], bytes))
- abi.encode(...) returns (bytes memory): ABI-encodes the given arguments
- abi.encodePacked(...) returns (bytes memory): Performs packed encoding of the given arguments. Note that packed encoding can be ambiguous!
- abi.encodeWithSelector(bytes4 selector, ...) returns (bytes memory): ABI-encodes the given arguments starting from the second and prepends the given four-byte selector
- abi.encodeWithSignature(string memory signature, ...) returns (bytes memory): Equivalent to abi.encodeWithSelector(bytes4(keccak256(bytes(signature))), ...)`

MEMBERS OF ADDRESS TYPES

<address>.balance (uint256):
balance of the Address in Wei

<address payable>.transfer(uint256 amount):
send given amount of Wei to Address, reverts on failure, forwards 2300 gas stipend, not adjustable

<address payable>.send(uint256 amount) returns (bool):
send given amount of Wei to Address, returns false on failure, forwards 2300 gas stipend, not adjustable

<address>.call(bytes memory) returns (bool, bytes memory):
issue low-level CALL with the given payload, returns success condition and return data, forwards all available gas, adjustable

<address>.delegatecall(bytes memory) returns (bool, bytes memory):
issue low-level DELEGATECALL with the given payload, returns success condition and return data, forwards all available gas, adjustable

<address>.staticcall(bytes memory) returns (bool, bytes memory):
issue low-level STATICCALL with the given payload, returns success condition and return data, forwards all available gas, adjustable