Lecture 4

Contracts and Functions

Solidity File Structure

Inside a .sol source file

- SPDX-License-Identifier
  - https://spdx.dev/
  - Can have an "unlicensed" identifier
- pragma
  - solidity version (to match compiler)
  - ABI encoder / decoder
  - Experimental pragmas: ABIv2, SMTchecker (Formal Verification)
- Import
- Comments
  - Single line: //
  - Multi line: /* */
  - Natspec

npm version semantic

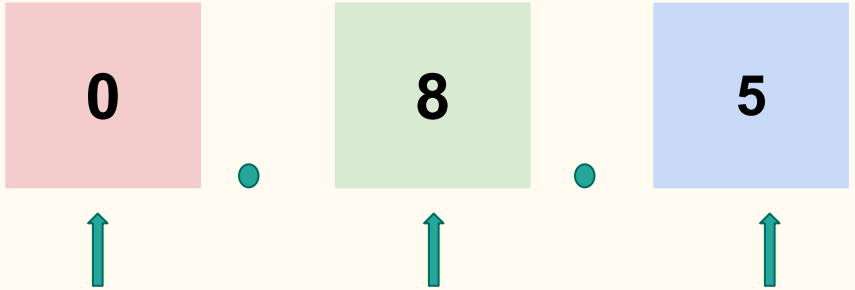
Solidity has not even hit 1 stable release yet!!!

^

use this specific version

>= <

range of versions to use



Importing

Virtual Filesystem on the Compiler

Initial files (plus dependencies) can be loaded on CLI or JSON format.

Compiler can add other files during compile time

Direct Import

```
import "/project/lib/util.sol";

import "lib/util.sol";

import "@openzeppelin/address.sol";

import "https://example.com/token.sol";
```

Relative Import

```
import "./";

import "../";
```

Natspec - Natural Language Specification Format

Tag		Context
`@title`	A title that should describe the contract/interface	contract, library, interface
`@author`	The name of the author	contract, library, interface
`@notice`	Explain to an end user what this does	contract, library, interface, function, public state variable, event
`@dev`	Explain to a developer any extra details	contract, library, interface, function, state variable, event
`@param`	Documents a parameter just like in Doxygen (must be followed by parameter name)	function, event
`@return`	Documents the return variables of a contract’s function	function, public state variable
`@inheritdoc`	Copies all missing tags from the base function (must be followed by the contract name)	function, public state variable
`@custom:...`	Custom tag, semantics is application-defined	everywhere

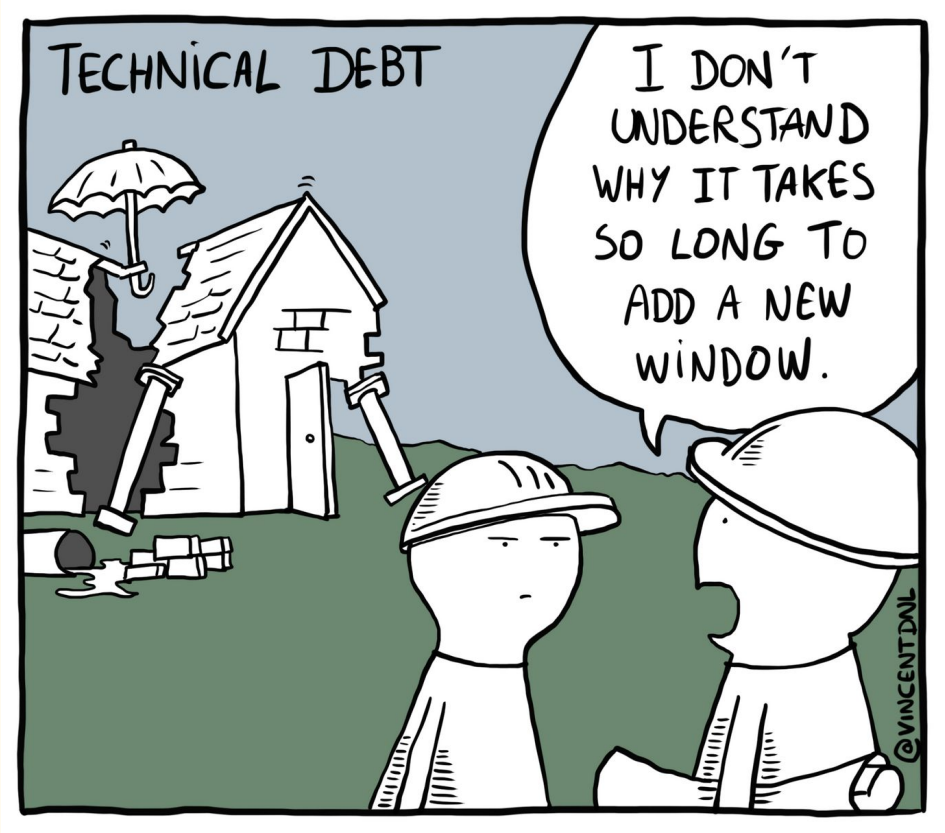
Natspec - An Example

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 < 0.9.0;

/// @title A simulator for trees
/// @author Larry A. Gardner
/// @notice You can use this contract for only the most basic simulation
/// @dev All function calls are currently implemented without side effects
/// @custom:experimental This is an experimental contract.
contract Tree {
    /// @notice Calculate tree age in years, rounded up, for live trees
    /// @dev The Alexandr N. Tetearing algorithm could increase precision
    /// @param rings The number of rings from dendrochronological sample
    /// @return Age in years, rounded up for partial years
    function age(uint256 rings) external virtual pure returns (uint256) {
        return rings + 1;
    }

    /// @notice Returns the amount of leaves the tree has.
    /// @dev Returns only a fixed number.
    function leaves() external virtual pure returns(uint256) {
        return 2;
    }
}
```

Source:

https://docs.soliditylang.org/en/v0.8.17/natspec-format.html

Technical debt - the cost of bad code



Solidity Conventions

Max Line Length = 120 char
Breakdown new lines uses tabs
	`thisFunctionCallIsReallyLong(`
	`  longArgument1,`
	`  longArgument2,`
	`  longArgument3`
	`);`

Encoding	UTF-8 or ASCII

Import Statements
Always at top after license identifier and pragma
	`// SPDX-License-Identifier: MIT`
	`pragma solidity >=0.4.0 <0.9.0;`
	
	`import "./Owned.sol";`

Whitespace
No space between brackets/quotes
Space around operators
	`spam(ham[1], Coin({name: "ham"}));`
	
	`x = 1;`
	`y = 2;`

Solidity Conventions - Naming

Contracts, Libraries, Interfaces, Structs, Events	`CapWords`
	
	`contract MyContract{}`
	`Struct PersonStruct{}`

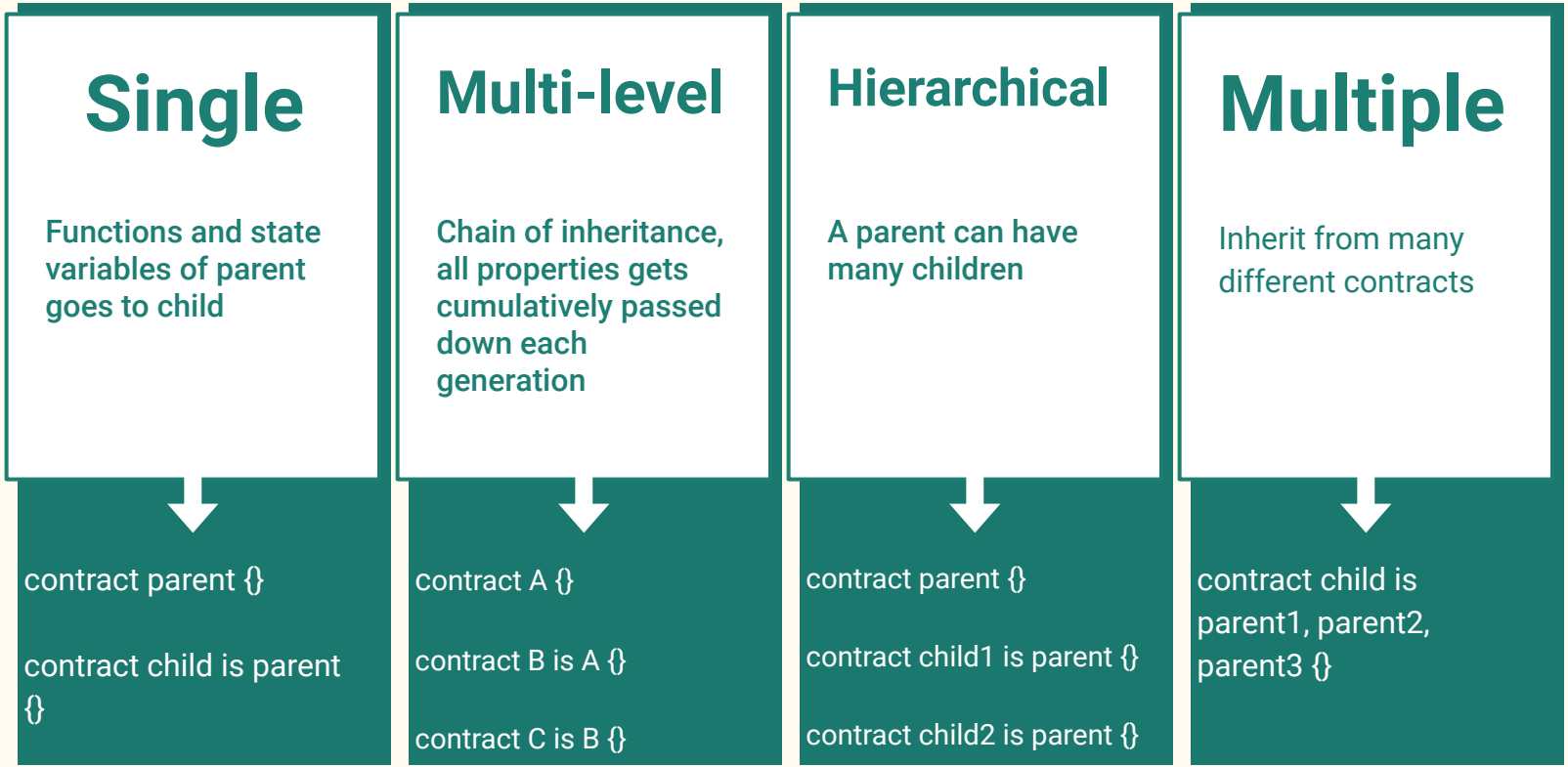
Function Names, Function Arguments, Variable Names	`mixedCase`
	
	`int myInteger;`
	`function helloWorld();`

Constants	`ALLCAPS`
	
	`int WINNING_NUMBER = 5;`

What's in a Smart Contract?

Inheritance and Memory

Inheritance - sharing is caring



Inheritance - Interfaces and Abstract Contracts

Interface

- Only used as templates
- No function implementation logic
- No constructors or state variables
- Only external functions, automatically virtual
- Cannot inherit

Abstract Contract

- Used as base for other contracts. is itself a contract
- Mix of implemented and unimplemented functions
- Can have constructor and state variable
- can inherit

```
interface InheritMe{
        function getBalance(address _user) external view{}
}
```

Interfaces - Birds of a feather, work together

Variable Scope in Smart Contracts

```
contract MyContract {
        int public a;
        mapping(address => int) ownerBalances;

        function checkBalance(address _owner) external view {
                int b = 5;
                return b + ownerBalances[_owner];
        }
}
```

- ➔ State variables are always stored onchain.
- ➔ They are inherited

- ➔ Local variables are in transient memory.
- ➔ They are not inherited

Memory Management in Smart Contracts

Memory

- Lasts for the length of a function call
- Stored on EVM
- Values are mutable
- Most versatile

Calldata

- Extremely tiny
- User inputs
- Immutable
- Use as safe guard

Storage

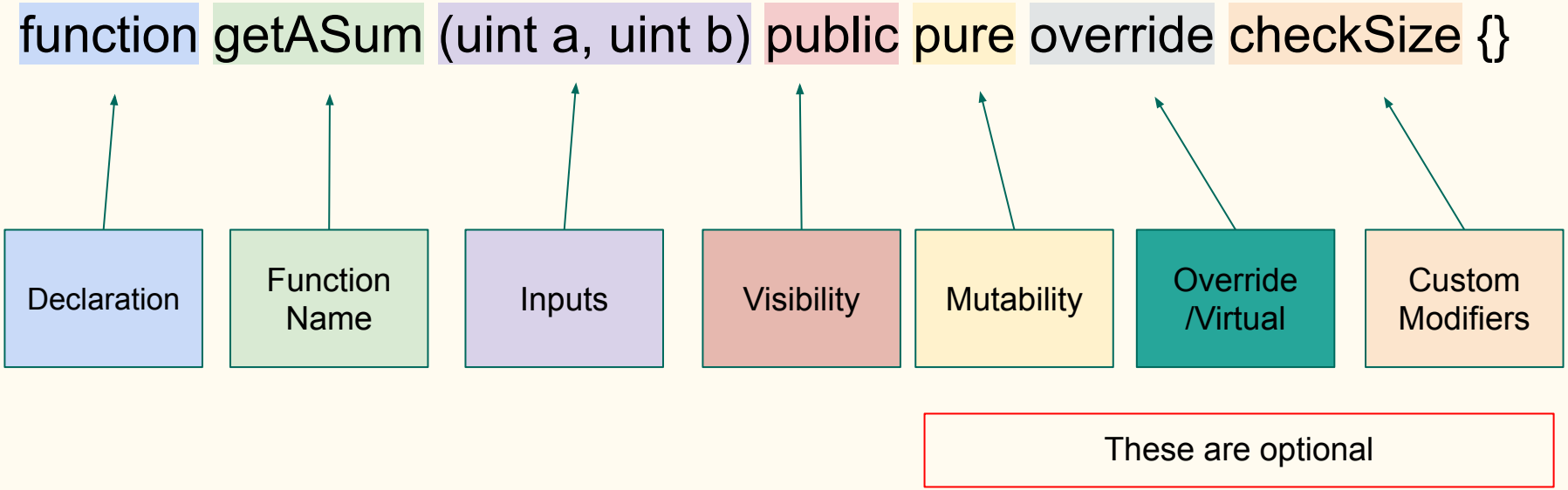
- Permanent on chain storage
- Extremely expensive!!
- Can use push function
- Use for state variables only

Stack

- Very small
- Stores things for immediate execution
- Gas Efficient
- **Extremely painful** -> inline assembly

Function Deep Dive

Function Convention



Function Visibility

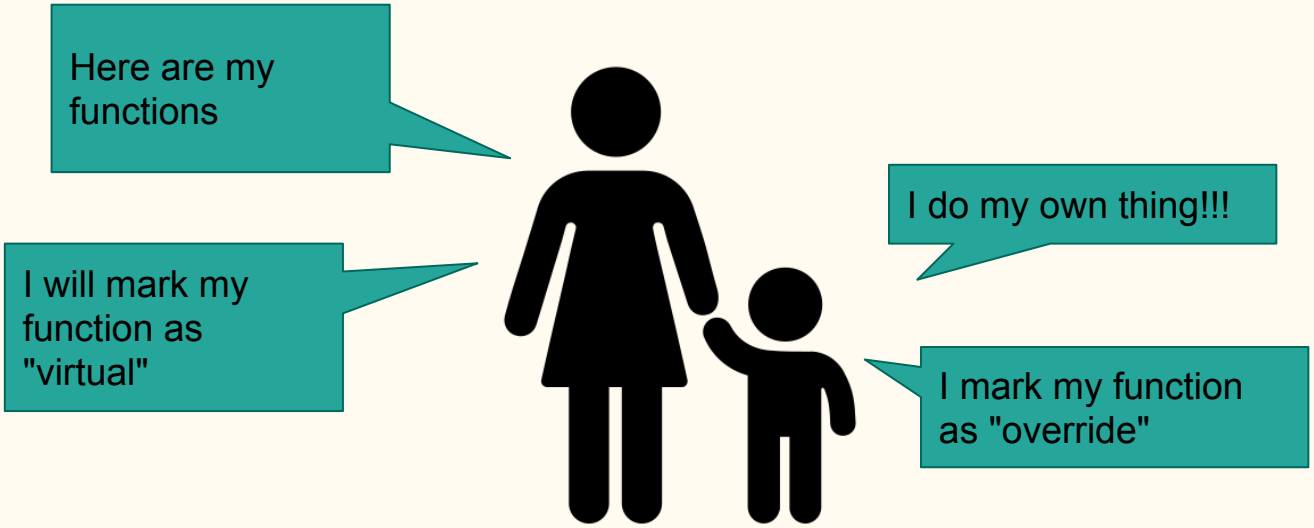
public	**private**
Function is callable by anyone, including other contracts	Function can only be executed within its contract. Cannot be inherited or externally called.
internal	**external**
Function can only be called by its contract and the contracts which inherit the contract it lives in.	Function will only ever be called by users, not used in other contracts. It is cheaper than public functions

Function Mutability

pure	view
No state variable will be changed or read.	State variable will be read but not changed.

Combined with "external", the function is free!

Function inheritance - virtual / override



Interface Functions are automatically virtual

Private Functions cannot be inherited

In the case of **Multiple Inheritance**, with parents who have functions with the same name, child contract MUST override the function with the same name!

Custom Function Modifiers

```
pragma solidity ^0.5.0;

contract Owner {
    address owner;
    constructor() public {
        owner = msg.sender;
    }
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
    modifier costs(uint price) {
        if (msg.value >= price) {
            _;
        }
    }
```

Declare using the "**modifier**" keyword
_; signals start of main function

Modifiers properties:

- Can be inherited
- Can take arguments
- Many in same function
- Can be overriden

Bonus: Function Overloading

```
function sameName(int a){}

function sameName(int a, string b){}

function sameName(int a, int b){}
```

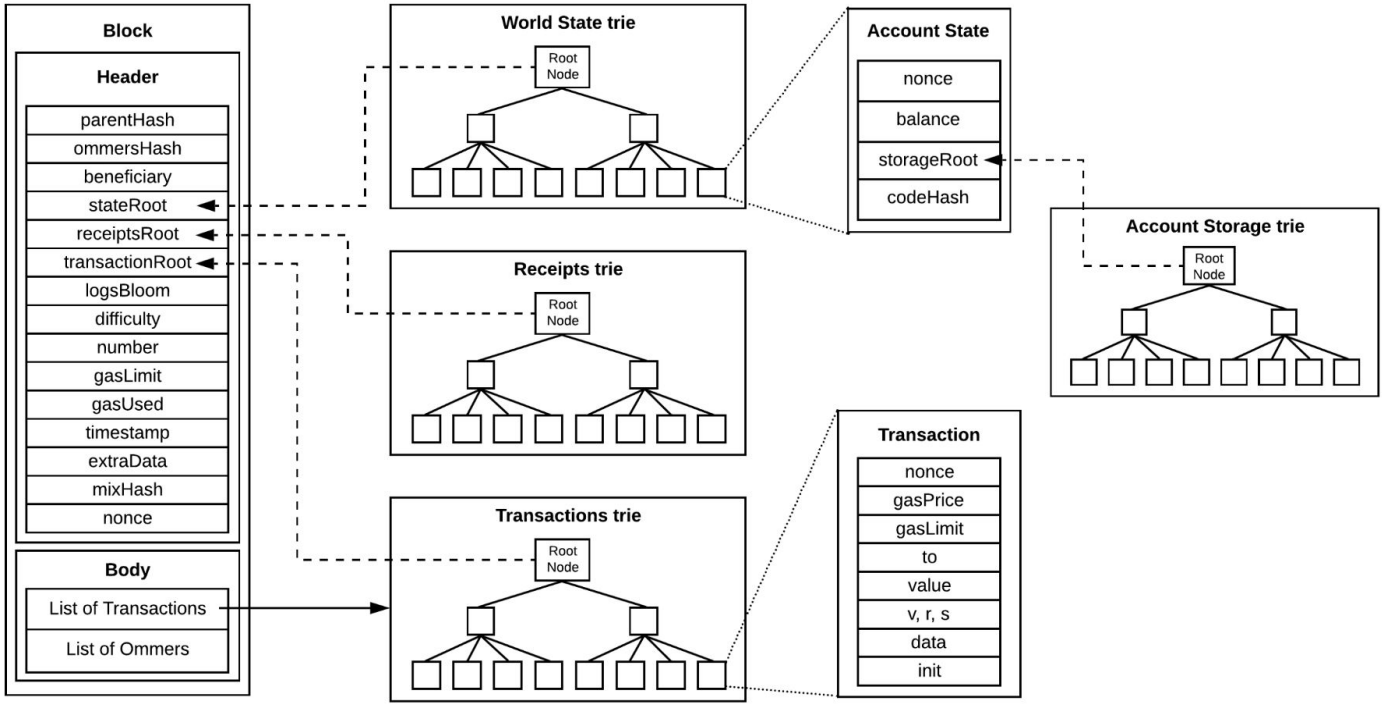
A function with multiple, different inputs but the same function name is an overloaded function.

Functions may be overload with:

- A different number of inputs
- Different types of inputs

Solidity Events

Ethereum Accounting - Tries Revisited



Block, transaction, account state objects and Ethereum tries

Events - Transaction Record

- Security Audits
  - Traditional accounting - does the stored variable add up to all events?

- User Interface
  - Confirm a transaction has happened
  - Change the interface after confirmation

- Cryptographic checks
  - Bloom filters

Event - Declaration and Actualization

```
contract Transaction {
    event makeATransfer(address indexed _from, address indexed _to, uint amount);

    function payRent(address receiver, uint deposit) external {
        require(msg.sender.balance >= msg.value);
        emit makeATransfer(msg.sender, receiver, amount);
    }
}
```

Events - ABI representation


```
{
        "returnValues": {
                "_from": "0x1111...FFFFCCCC",
                "_to": "0x50...sd5adb20",
                 "amount": "0x420042"
        },
        "raw": {
                "data": "0x7f...91385",
                "topics": ["0xfd4...b4ead7", "0x7f...1a91385","0xf28...d21297" ]
        }
}
```