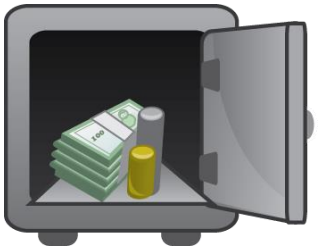
Building a Beginner Smart Contract

A Little About Me...

- Software Engineer @ Microsoft in Pittsburgh
- Run https://chaintuts.com creating cryptocurrency & blockchain related tutorials
- Articles, videos, and code projects
- On YouTube, Twitter, Github
- Support: Patreon, Crypto, Spreadshirt Apparel
- Focus is on understanding & teaching core concepts

First, A Quick Disclaimer

- I’m not an expert in smart contract dev…not even intermediate
- I am a professional software engineer!
- Want to give my perspective on building a beginner smart contract…as a beginner!



Programming 101

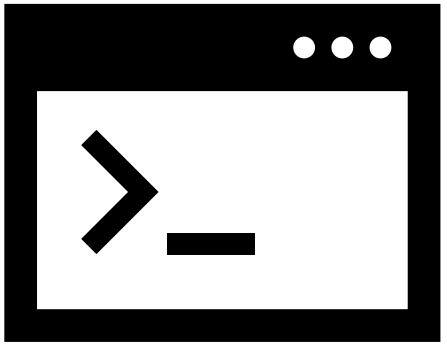
- Devices don’t do anything on their own…
- We have to “program” them with explicit instructions
- Laptop, phone, even your car
  - Instructions interpreted by a CPU…make it do something
  - Calculations, display, accept inputs, etc.
- Do I have to learn machine code for every device??
  - No, because of *abstractions*

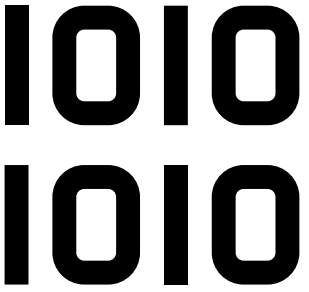
Abstractions

- We've built layers between machine code and the "high level" languages we use
- Don't have to know x86 assembly to code on a laptop
- Can learn C, C++, Python, Java, etc. which are all "translated" to the machine you're on

Types of Languages

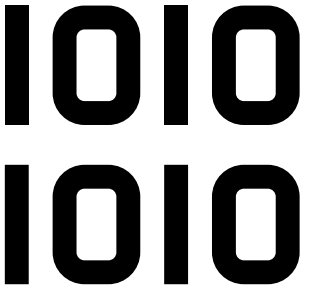
- Compiled
- Interpreted
- Intermediate (bytecode)





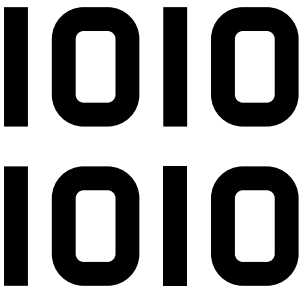
Types of Languages

- Compiled
  - Ex: C, C++
  - Instructions are read by a “compiler” -> translated directly into machine code for the *target platform*
  - Have to re-compile for each target platform/OS
    - Ex: Linux, ARM (Raspberry Pi)
    - Ex: Windows, x86-64



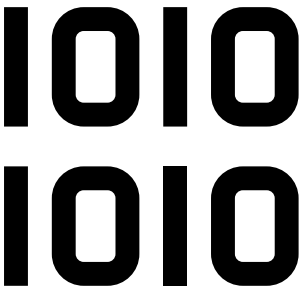
Types of Languages

- Interpreted
  - Ex: Python, JavaScript
  - Instructions are read and then executed line-by-line
    - Interpreter translates instructions to machine/system calls on the fly
  - No need for compilation, just need interpreter available on that system!
    - Ex: Python on Windows, Linux, CircuitPython devices
    - Ex: JavaScript in your web browser



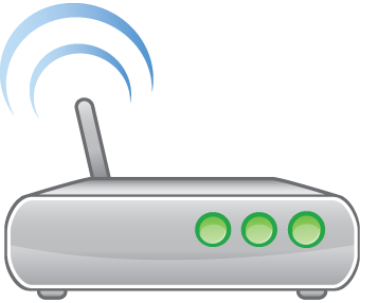
Types of Languages

- Intermediate Languages
  - Ex: Java, C#.NET, *Solidity*
  - Instructions are compiled into an *intermediate language* called “bytecode”
  - The language *virtual machine* then interprets/recompiles instructions to the target machine – various strategies for this
  - Ex:
    - Java compiled on a Windows, executing on an Android Phone
    - C#.NET compiled on Windows, running on an XBOX360



Programmable Blockchains

- What does this have to do with Blockchains?
  - Many blockchains are programmable!
  - Bitcoin and Bitcoin Cash use an interpreted language called Script for transactions
    - Allows for legacy, multisig, segwit (BTC), SLP tokens (BCH), etc.
  - Ethereum is even more programmable…

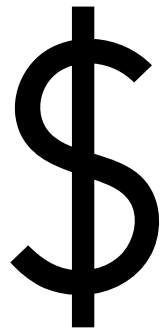


The Ethereum Virtual Machine

- Ethereum features an intermediate language for running smart contracts (Ethereum Bytecode)
- We can write contract code using different languages (Solidity, Vyper) and compile to bytecode for execution
- There are some key differences between coding in say…Java…and using the EVM

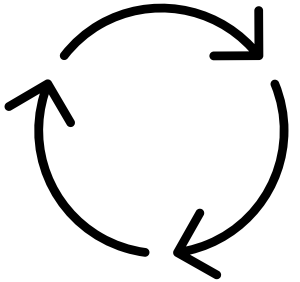
PC vs. Ethereum

- Cost:
  - Execution on a PC is CHEAP!
  - Every operation on Ethereum costs “gas”, a unit for measuring compute resources used by executing the contract
  - Don’t use Ethereum for compute-heavy, performant code. Go rent an Azure VM for that



PC vs. Ethereum

- Turing-completeness
  - Abstract concept, but think of the ability to use loops, etc. in code
  - Bitcoin Script is non-turing complete – cannot loop forever
  - Ethereum IS turing complete…so not a difference but noteworthy
  - Prevent locking up networking resources with gas limit – can’t loop infinitely



PC vs. Ethereum

- Determinism
  - PC program may execute differently every time on a PC
    - Ex: Computer game with some random elements
  - Smart contract will *always* execute the same on any network node!
    - Must be this way to have a decentralized network
    - Any node/miner can execute the contract transaction and verify the end result

Building a Smart Contract

- The idea: Implement a basic dice roll contract
- Call a contract function, get back a return value 1-6
- Just need a basic contract for this – no state, etc.
- Use it for something fun…picking backyard bike park features for practice ☺

The Code

```
pragma solidity ^0.6.6;


contract BikeParkDice
{
    // The only function for this contract
    // It uses some seed data and a hash to return a "random" number from 1-6
    function rollDice() public view returns(uint256 final_roll)
    {
```

The Code

```
 // Get the sending address, block difficulty, and block timestamp
        // to give us a good enough "seed" for our pseudo-random number
        bytes32 seed = keccak256(abi.encodePacked(msg.sender));
        bytes32 seed_2 = keccak256(abi.encodePacked(block.difficulty));
        bytes32 seed_3 = keccak256(abi.encodePacked(block.timestamp));
```

The Code

```
 // Hash the seed data. This keccak hash is deterministic (necessary for Ethereum contracts!)
        // and is preimage resistant, which is helpful for our pseudo-random generation
        bytes32 hash = keccak256(abi.encodePacked(seed, seed_2, seed_3));


        // Cast the hash bytes to an integer value
        uint256 hash_num = uint256(hash);
```

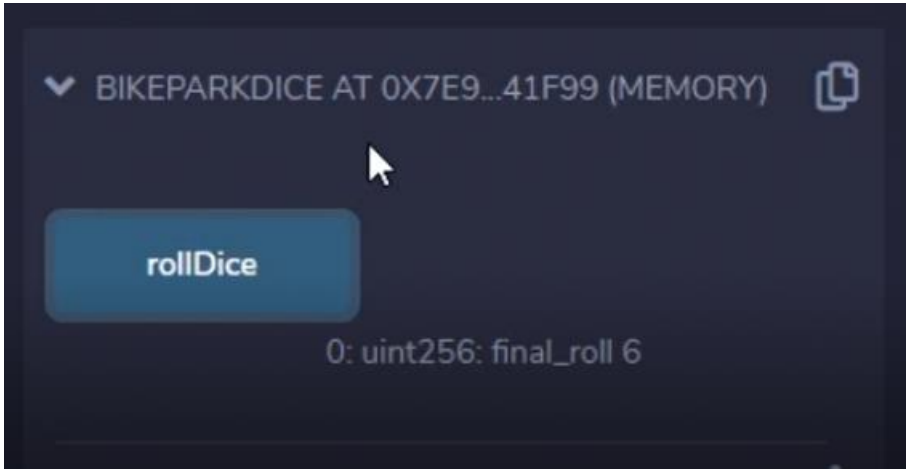
The Code

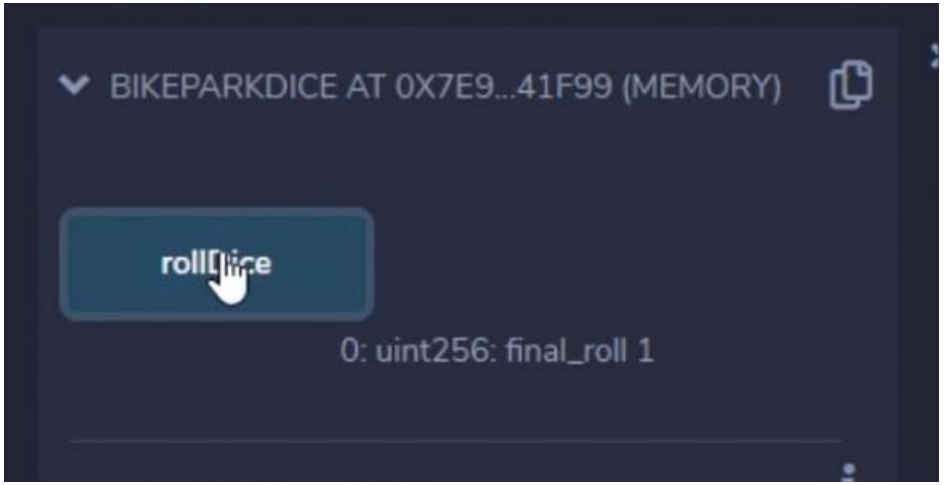
```
 // Return a number from 1-6 by using the modulo (remainder) function
        final_roll = (hash_num % 6) + 1;
        return final_roll;


    }
}
```

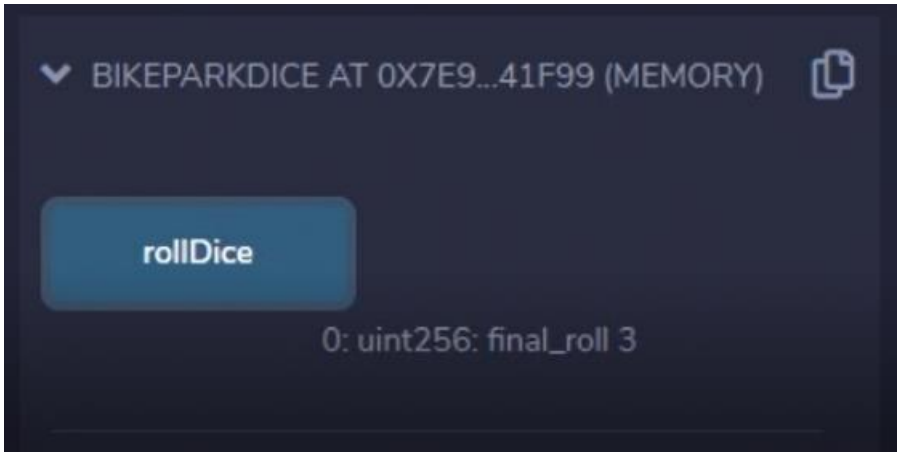
Development and Testing

- Used Remix IDE at https://remix.Ethereum.org
- Built in Solidity Compiler and JavaScript VM for testing!
  - Didn’t need to deploy contract to real testnet or mainnet to experiment with it.
  - Could easily call function and test return value
    - View function – called on a local node using API such as web3.js – no state change
    - Functions that change state must be called via transaction and “watched” for return

Development and Testing







Real Deployment

- Beyond just testing – How would I deploy?
- Compile Solidity contract down to bytecode
- Create transaction with bytecode as data and send to 0x0…0 (the “zero address”)
- Wait for contract to be mined
- Contract will live at a contract address, used to send new transactions with calls in them

Ethereum Contracts

- Use a virtual machine/intermediate language compiled from HLL like Solidity
- Are Turing complete, but require gas
- Are deterministic (needed for decentralization)
- Use where decentralization is key!
  - Representing real-world assets (Non-fungible tokens), creating new currencies (tokens), legal applications, decentralized finance, etc.

Questions?