Introduction to Crypto Zombies

Brought to you by Wei Ren

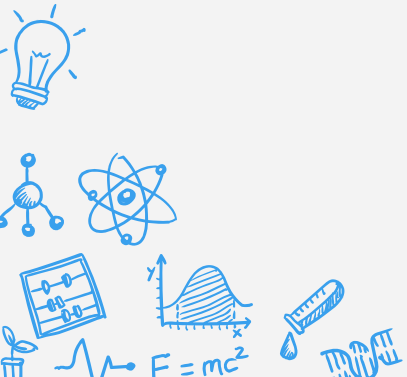
What is CryptoZombies?

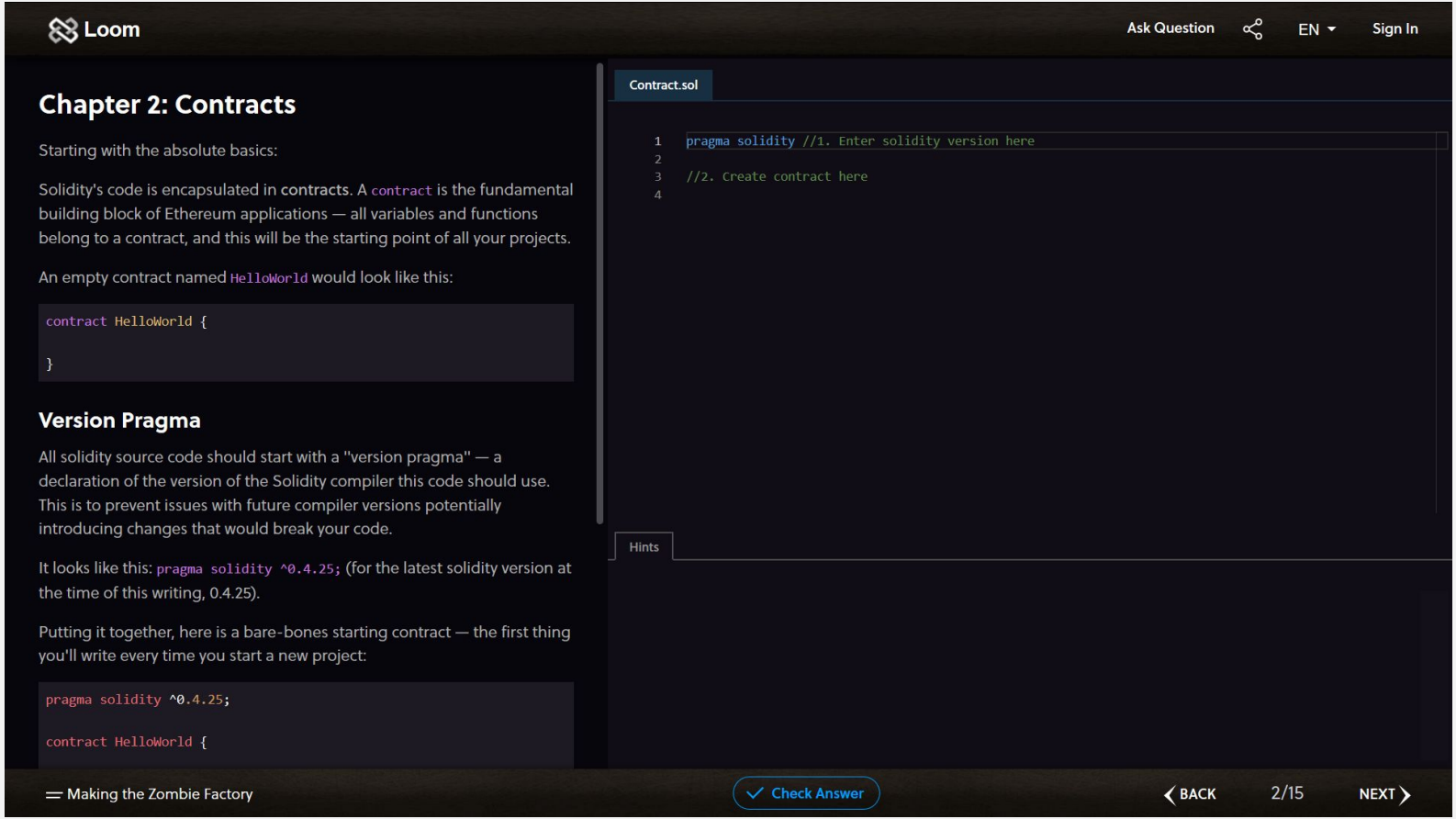
- At the end of the sharing session, you will be able to:

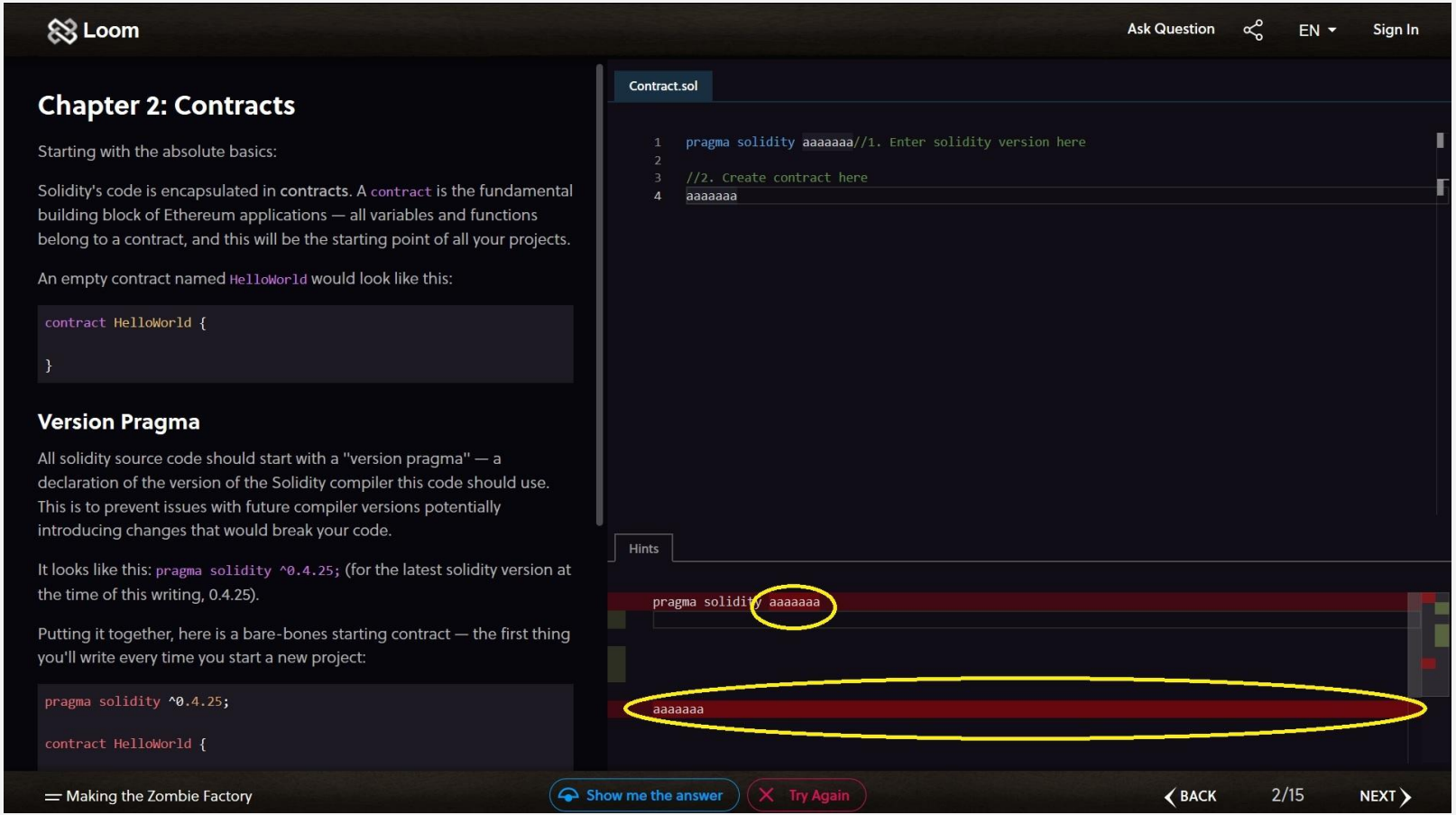
❑ Create a Zombie Factory
❑ Customise your own zombies
❑ Attack each other’s zombies
❑ Feed your own zombies to level up

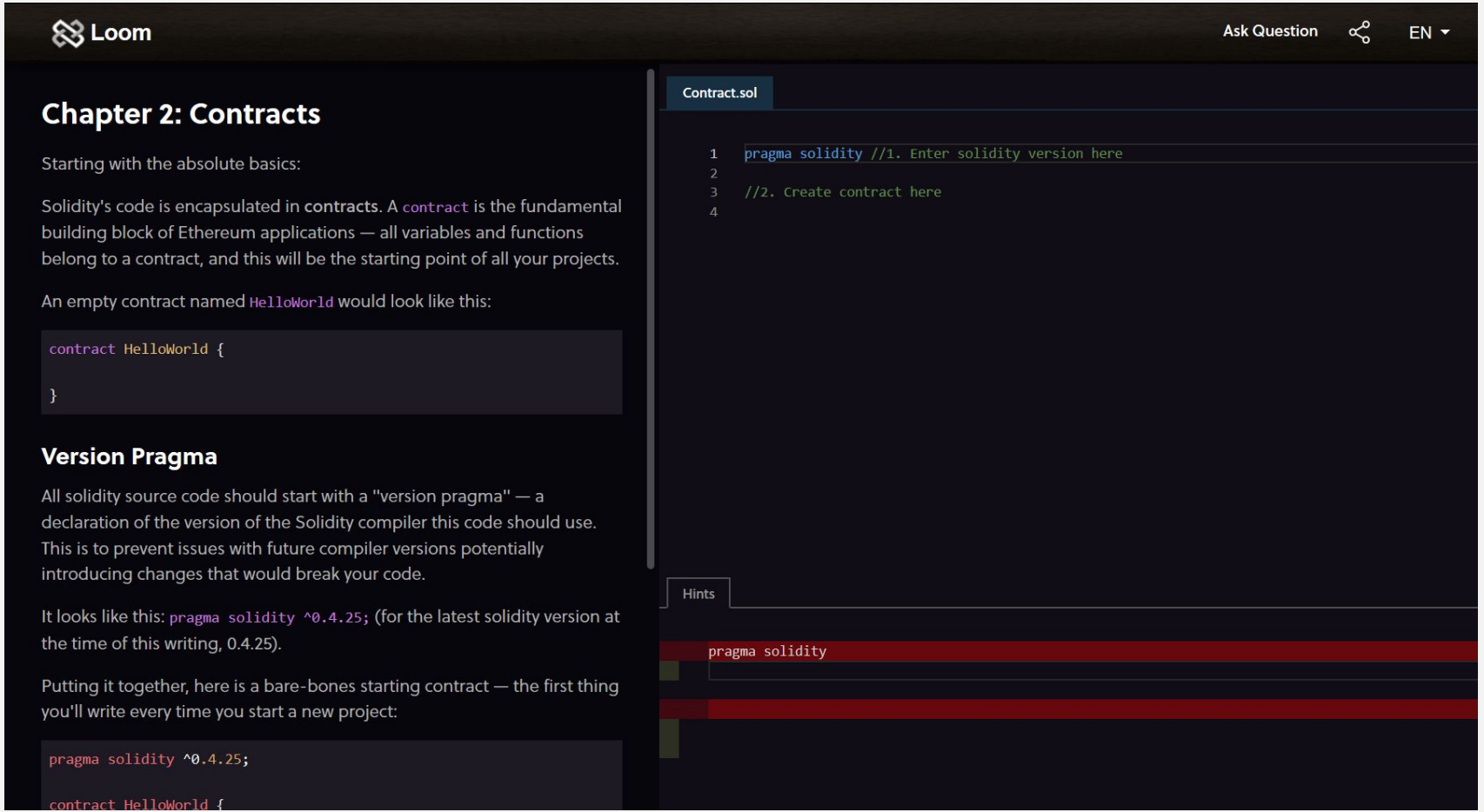
Preparation

https://cryptozombies.io/



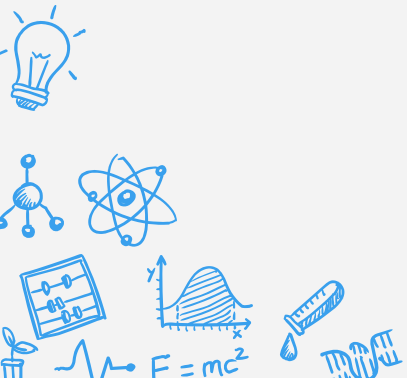

Loom
Ask Question
EN
Sign In
Chapter 2: Contracts
Starting with the absolute basics:
Solidity's code is encapsulated in contracts. A contract is the fundamental building block of Ethereum applications — all variables and functions belong to a contract, and this will be the starting point of all your projects.
An empty contract named HelloWorld would look like this:
contract HelloWorld {
}
Version Pragma
All solidity source code should start with a "version pragma" — a declaration of the version of the Solidity compiler this code should use. This is to prevent issues with future compiler versions potentially introducing changes that would break your code.
It looks like this: pragma solidity ^0.4.25; (for the latest solidity version at the time of this writing, 0.4.25).
Putting it together, here is a bare-bones starting contract — the first thing you'll write every time you start a new project:
pragma solidity ^0.4.25;
contract HelloWorld {
Contract.sol
1 pragma solidity //1. Enter solidity version here
2
3 //2. Create contract here
4
Hints
Making the Zombie Factory
Check Answer
BACK
2/15
NEXT





Lesson 1 : Creating a Zombie Factory

1. Instantiate new zombies
2. Produce random zombies dna
3. Assign each created zombie with a random dna



Lesson 1 : Creating a Zombie Factory

```
pragma solidity ^0.4.25;


contract ContractName{
    //Declare variables
    //Implement functions
}
```

Characteristics of Zombies

Zombie DNA(16 integers):

44 | 24 | 47 | 521 | 258 | 4128

Head | Eye | Shirt | Skin Color | Eye Color | Clothes

Data Types

Unsigned Integers: uint256/uint, uint8

Strings : string

Structures: struct

Arrays: Objects[] public *arrayname*

Address

Function declaration

- Naming Convention: camelCase
- Syntax:

function *functionname(datatype _parameter1, …)*
private/public pure/view returns (datatype ...)

E.g. function _sayNumber(uint _num) private pure returns(string){....}

Function Visibility

- Private
  - ↳ Callable from other functions in the contract
- Public
  - ↳ Callable from anywhere
- Internal
  - ↳ Like private + Callable from from functions in derived contracts
- External
  - ↳ Callable only from outside of the contract

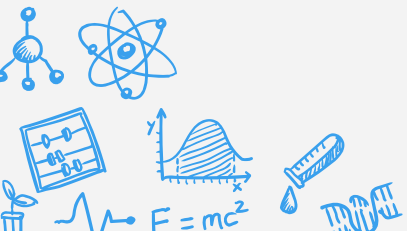
Function Modifiers - State Modifiers

- Pure
  - No modifications + reading of data is done
- View
  - No modifications to the data is done
  - Reading data from the blockchain

Events

- A way for the contract to inform the app's front end that something has occurred on the blockchain.

1. Declare the event in the contract
2. Fire the event within some of the functions when required



Events

```
contract ZombieFactory{
    event NewZombie(uint zombieId, string name, uint dna); //Declare the event in the contract
    ...
function createNewZombie(string _name, uint _dna) private {
    //Some logic
    uint id= ... ;
    emit NewZombie(id, _name, _dna) //Fire the event
}
}
```

Keccak256 & abi.encodePacked()

keccak256() : An in-built hash function which generates 256bit hexadecimal value (ZombieDna)

- Only takes in 1 parameter with a type bytes

abi.encodePacked(): Converts a data type to bytes

Lesson 2: Zombie Feeding

```
pragma solidity ^0.4.25;

import “./ContractA”;


contract ContractB is ContractA{

        …

}
```

Inheritance

- Contract B will have access to *some* functions of Contract A
- private functions will not be visible in Contract A
- Used to break up long lines of code into multiple contracts

msg.sender

- Refers to the address of the person/smart that called the current function

Mappings

- A store for key-value pairs
- Syntax:

mapping (datatype => datatype) [public/private] mappingname

E.g.

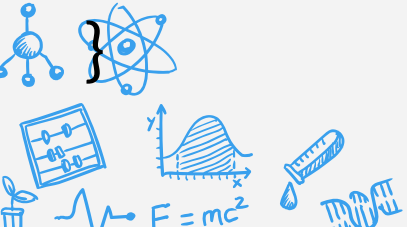
mapping (address => uint) ownerZombieCount;

- To access value: ownerZombieCount[msg.sender]

require

- Used to verify certain conditions before running the function
- Syntax:

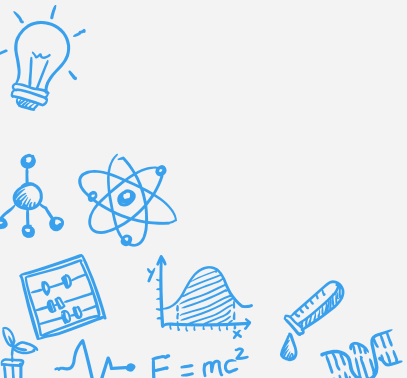
```
function MyFunction{
    require(condition);
    //Some logic
}
```



Storage vs Memory

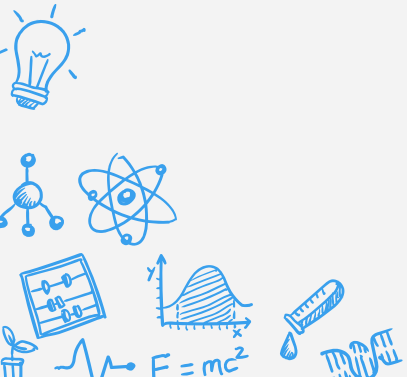
Memory : Temporary (local variables in functions)

Storage : Stores data permanently on the blockchain (State variables)



Lesson 3 : Advanced Solidity Concepts

- ❑ Ownable Contracts & Function modifiers
- ❑ Gas
- ❑ For Loops



Ownable Contract

- Taken from OpenZeppelin library
- Secure and Community-vetted smart contracts
- Link:

  https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/ownership/Ownable.sol

Function modifiers - Custom modifiers

```
29      /**
30       * @dev Throws if called by any account other than the owner.
31       */
32      modifier onlyOwner() {
33          require(isOwner());
34          _;
35      }
```

- Restricts access to certain functions
- Only the owner of the contract could access

Function modifier

```
function _createZombie(string _name, uint _dna) internal
onlyowner{

...

}
```

Saving Gas - Struct packing

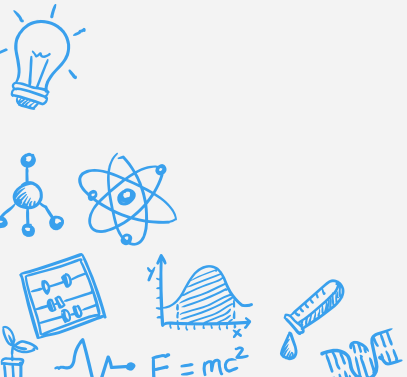
- For any uint state variable, 256 bits of storage will be reserved.
  - ↳ Thus, declaring a uint8/ uint16/ uint32 state variable will NOT save any gas
- Struct packing
  - ↳ Packing all uint variables into a struct to save on the storage space

Saving Gas - Memory

- Storing data in storage is expensive
  - ↳ Reason: Data is stored permanently on the blockchain
- Thus, an alternative way is to rebuild that data into memory (temporary)
  - ↳ Very useful in sorting data in storage arrays

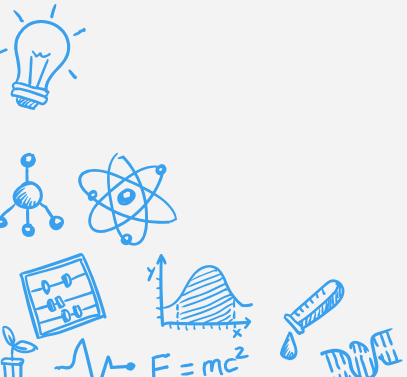
Saving Gas - View Functions

- External view functions incur no gas
  - However, internal view functions still incur gas since the calling function still creates a transaction which need to be verified by the nodes



Lesson 4: Zombie Battle System

- ❑ Payable



Function modifier - Payable modifier

- Allows us to call a function and pay Ether to the contract simultaneously
- E.g.

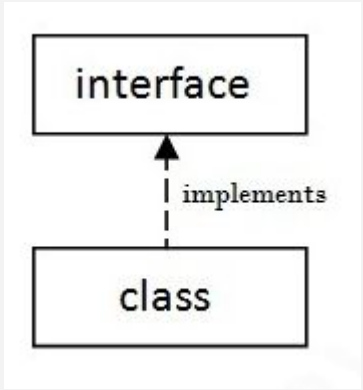
```
function buy() external payable{
    require(msg.value == 1 ether);
    //Some logic to transfer ownership
}
```

Part 2. Let's make more Zombies !

Brought to you by Kelvin

Ethereum Tokens ?

- A Token on Ethereum is a smart contract that adhere to a standard set of function which all token contracts share. (E.g. totalSupply, balanceOf…)
- Think of it like an “Interface” in programming.



Ethereum Tokens ?

- Within the smart contract itself, it contains a mapping which stores the balances of each address has.
- Hence, a token also keep a records of who owns how much of the token.

ERC 20

```
1  // ----------------------------------------------------------------------------
2  // ERC Token Standard #20 Interface
3  // https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
4  // ----------------------------------------------------------------------------
5  contract ERC20Interface {
6      function totalSupply() public view returns (uint);
7      function balanceOf(address tokenOwner) public view returns (uint balance);
8      function allowance(address tokenOwner, address spender) public view returns (uint remaining);
9      function transfer(address to, uint tokens) public returns (bool success);
10     function approve(address spender, uint tokens) public returns (bool success);
11     function transferFrom(address from, address to, uint tokens) public returns (bool success);
12
13     event Transfer(address indexed from, address indexed to, uint tokens);
14     event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
15 }
```

https://goo.gl/88kYXp

ERC 721

```
/// @title ERC-721 Non-Fungible Token Standard
/// @dev See https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md
///  Note: the ERC-165 identifier for this interface is 0x80ac58cd
interface ERC721 /* is ERC165 */ {
    event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);

    event Approval(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);

    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);

    function balanceOf(address _owner) external view returns (uint256);

    function ownerOf(uint256 _tokenId) external view returns (address);

    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;

    function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;

    function transferFrom(address _from, address _to, uint256 _tokenId) external payable;

    function approve(address _approved, uint256 _tokenId) external payable;

    function setApprovalForAll(address _operator, bool _approved) external;

    function getApproved(uint256 _tokenId) external view returns (address);

    function isApprovedForAll(address _owner, address _operator) external view returns (bool);
}
```

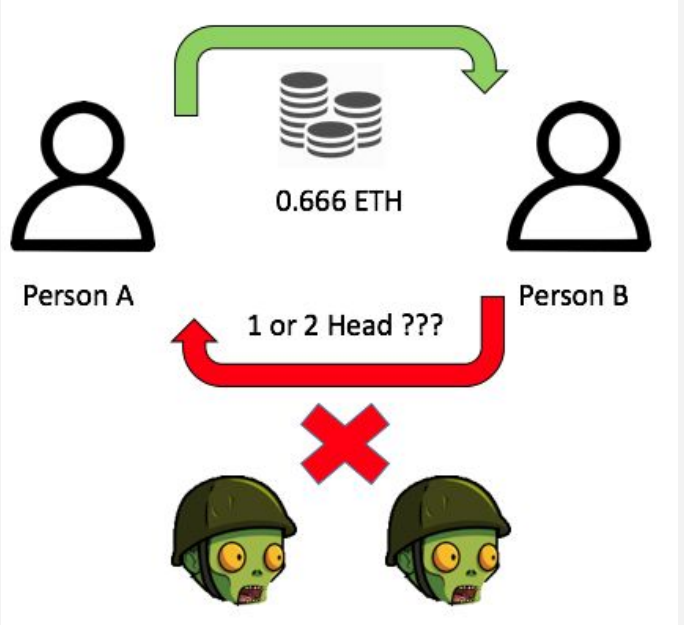
http://erc721.org/

Difference Between ERC 721 & 20

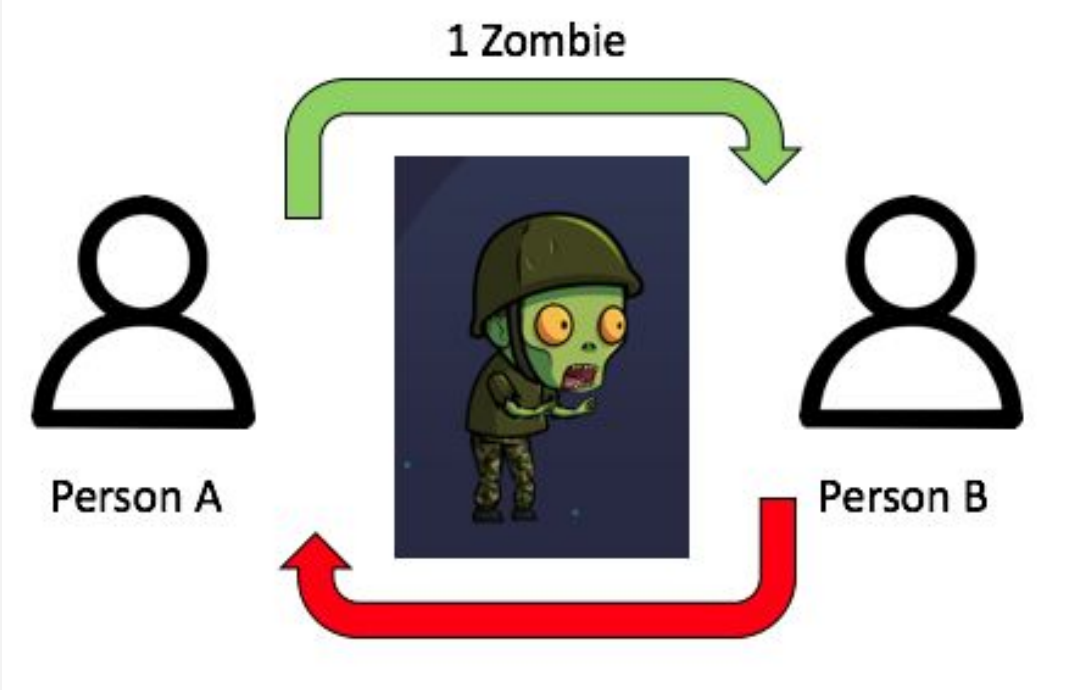
- ERC 20
  - Ethereum coin can be broken down into smaller portion such as 0.000001 (decimal places).
  - Unable to distinguish the coin, since every token is the same.
- ERC 721
  - Cannot be broken down into smaller portions. Traded as whole unit.
  - Every token is unique in its own.

Why it matters in CryptoZombies

- Tokens that act as currencies is not suitable for CryptoZombies.
- Not all zombie are equal.
  Level 1 Zombie is not equal to Level 35 Zombie



Why it matters in CryptoZombies



Why it matters in CryptoZombies

Contract Security; Overflow & Underflow

- Overflow & Underflow. What it means ?

		Number (uint8)								
Bits	256	128	64	32	16	8	4	2	1	Number
		0	0	0	0	0	0	0	0	0
		1	1	1	1	1	1	1	1	255
		1	1	1	1	1	1	1	1	255
		0	0	0	0	0	0	0	1	1
	1	0	0	0	0	0	0	0	0	256

Contract Security; Overflow & Underflow

- How to avoid this issue ?
  - OpenZeppelin is a library for secure smart contract development.
  - It has created a math library called SafeMath.
- How to use the library ?
  - `import "./safemath.sol";`

```
using SafeMath for uint256;

uint256 a = 5;
uint256 b = a.add(3); // 5 + 3 = 8
uint256 c = a.mul(2); // 5 * 2 = 10
```

Contract Security; Overflow & Underflow

- How does it overcome the issue?
  - E.g. Addition And Subtraction function in SafeMath

```
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
```

```
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }
```

Contract Security; Overflow & Underflow

- How does it overcome the issue?
  - Just like normal math arithmetic but with an addition of the assert statement.
  - E.g. Addition, it would check the sum of the 2 uint against ‘a’ ensuring that the sum is greater than ‘a’. This would prevent the sum from overflows.
  - Assert is similar to require, where it would return an error if the condition is false. However, assert would not refund the user the remaining gas when a function fails.

Contract Security; Overflow & Underflow

- When to use assert or require ?

Use `require()` to:

- Validate user inputs
- Validate the response from an external contract ie. use `require(external.send(amount))`
- Validate state conditions prior to executing state changing operations, for example in an `owned` contract situation
- Generally, you should use `require` more often,
- Generally, it will be used towards the beginning of a function.

Use `assert()` to:

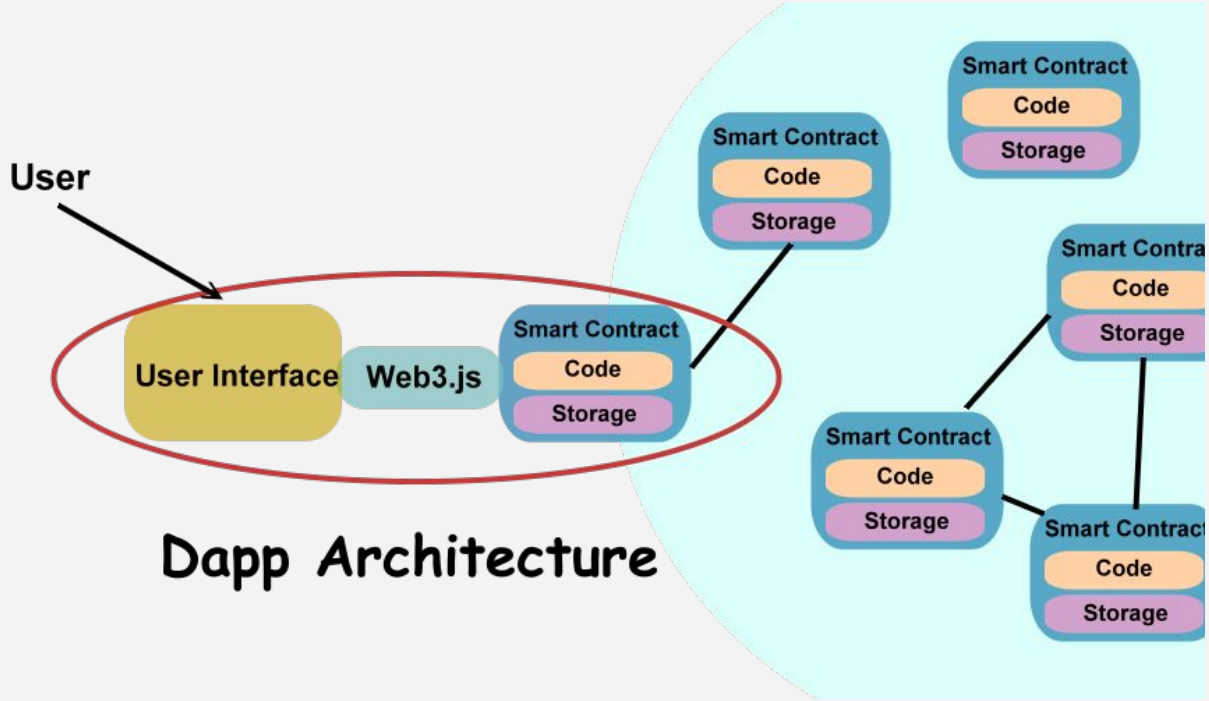
- check for overflow/underflow
- check invariants
- validate contract state *after* making changes
- avoid conditions which should never, ever be possible.
- Generally, you should use `assert` less often
- Generally, it will be use towards the end of your function.

Basically, `assert` is just there to prevent anything really bad from happening, but it shouldn't be possible for the condition to evaluate to false.

Web3.js

- What is Web3.js
  - It a Javascript library which allow user to interact with a ethereum node, using a HTTP connection.
    (Interaction - query the function within the smart contract by providing the following items)
    - Address of the smart contract.
    - Function name that the user want to call.
    - Variables that user want to pass into the function.

Web3.js



https://goo.gl/d5hm7p

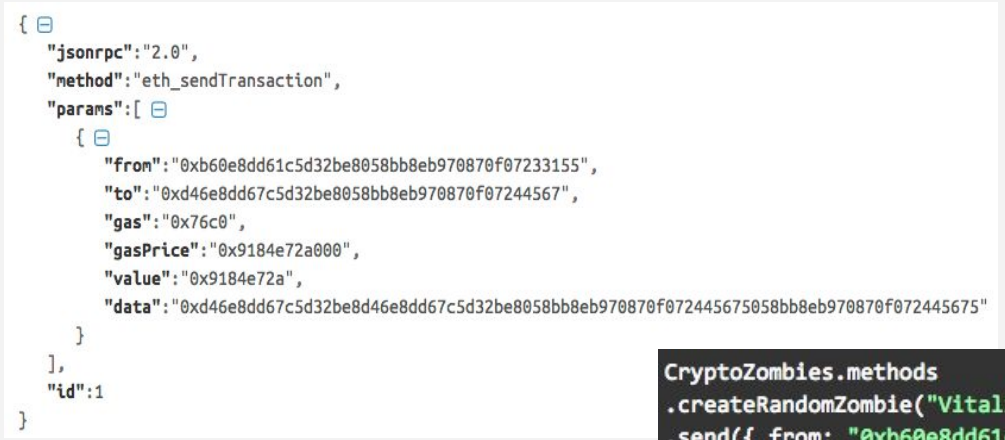
Web3.js

- How to use Web3.js
  - Since it is a Javascript library, you can import the library like any javascript library in html by specifying the script within the <head></head> tag in html.
  -

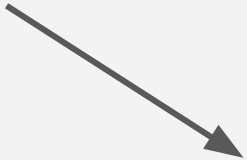
```
<head>
  <meta charset="UTF-8">
  <title>CryptoZombies front-end</title>
  <script language="javascript" type="text/javascript"
src="https:cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></scr
ipt>
  <!-- Include web3.js here -->
  <script language="javascript" type="text/javascript"
src="web3.min.js"></script>
</head>
```

Why the need for Web3.js

- Simplify the need for creating complex queries in JSON-RPC.



```
{
  "jsonrpc":"2.0",
  "method":"eth_sendTransaction",
  "params":[
    {
      "from":"0xb60e8dd61c5d32be8058bb8eb970870f07233155",
      "to":"0xd46e8dd67c5d32be8058bb8eb970870f07244567",
      "gas":"0x76c0",
      "gasPrice":"0x9184e72a000",
      "value":"0x9184e72a",
      "data":"0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675"
    }
  ],
  "id":1
}
```

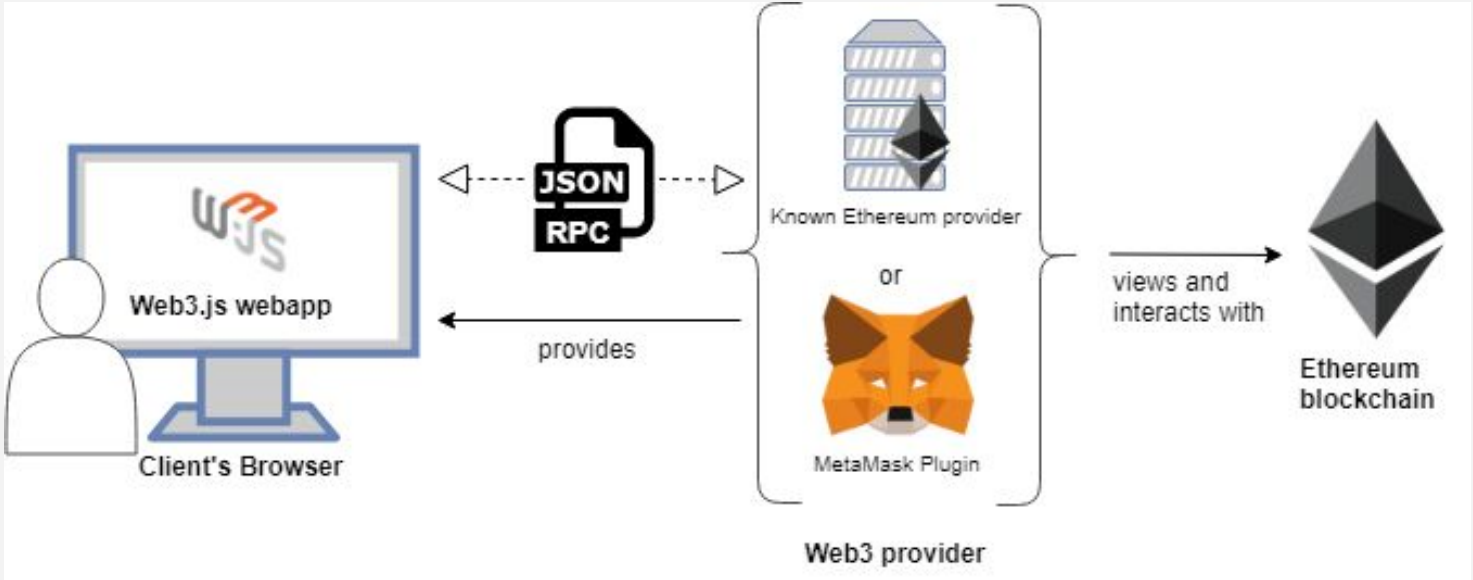


```
CryptoZombies.methods
.createRandomZombie("Vitalik Nakamoto 🤔")
.send({ from: "0xb60e8dd61c5d32be8058bb8eb970870f07233155", gas: "3000000"
})
```

Web3 Provider & Metamask

- A Web3 provider handle the incoming read and write requests from the user and direct it to the specific node that the user want to communicate with.
- Metamask manage Ethereum accounts and keys which these account is use to interact with website running with Web3.js. (Write & Read to the blockchain. Including signing transaction with Private key)

Web3 Provider & Metamask



https://goo.gl/veKmtX

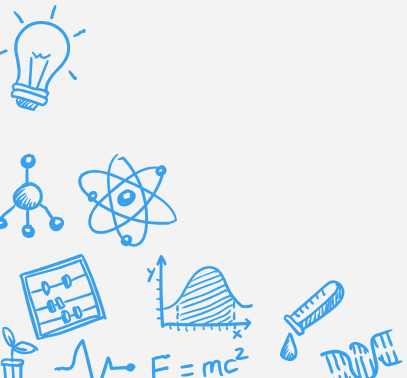
Using Metamask's web3 provider.

```
window.addEventListener('load', function() {

  // Checking if Web3 has been injected by
  //   the browser (Mist/MetaMask)
  if (typeof web3 !== 'undefined') {
    // Use Mist/MetaMask's provider
    web3js = new Web3(web3.currentProvider);
  } else {
    // Handle the case where the user doesn't
    // have web3. Probably show them a message
    // telling them to install Metamask in
    // order to use our app.
  }

  // Now you can start your app & access web3js
  // freely:
  startApp()

})
```

- Add this boilerplate code to the <script> tag within the <body>

Communicate With Contracts

- 2 Things required to communicate with contracts
  - Contract Address.
  - ABI (Application Binary Interface).



Calling Contract Functions

- Web3.js has 2 methods to call the function on the contracts.
  - Call (View & Pure function)
    - Read Only.
    - Do not incur any gas.
  - Send (Any other functions other than View & Pure)
    - Create transaction.
    - Change data on the blockchain.
    - Require to pay gas and signing of the transaction with the user's private key.
  - Syntax :

```
//Call Syntax
myContract.methods.myMethod(123).call()

//Send Syntax
myContract.methods.myMethod(123).send()
```

Calling Contract Functions 2

```
function getZombieDetails(id) {
  return cryptoZombies.methods.zombies(id).call()
}

// 1. Define `zombieToOwner` here
function zombieToOwner(id) {
  return cryptoZombies.methods [illegible]
}

// 2. Define `getZombiesByOwner` here
function getZombiesByOwner(owner) {
  return cryptoZombies.methods [illegible]
}
```

- Similar to the structure as getZombieDetails(id)

ZombieHelper.sol

```
function getZombiesByOwner(address _owner) external view returns(uint[]) {
  uint[] memory result = new uint[](ownerZombieCount[_owner]);
  uint counter = 0;
  for (uint i = 0; i < zombies.length; i++) {
```

ZombieFactory.sol

```
Zombie[] public zombies;

mapping (uint => address) public zombieToOwner;
mapping (address => uint) ownerZombieCount;
```

Metamask & Account

- MetaMask allow user to manage multiple accounts.
- How to get the current active account ?

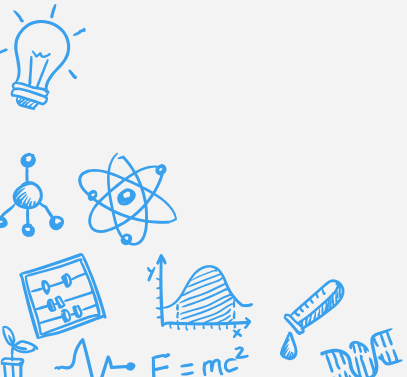
```
var account = web3.eth.accounts[0];
var accountInterval = setInterval(function() {
  if (web3.eth.accounts[0] !== account) {
    account = web3.eth.accounts[0];
    updateInterface();
  }
}, 100);
```

- Check every 100 milliseconds if userAccount is still equal web3.eth.accounts[0.]
- If not, reassigns userAccount to the currently active account, and calls a function to update the display.

https://goo.gl/FYMUzW

Sending Transaction

- Requires a from address of who's calling the function. (Msg.sender)
- Cost gas.
- A significant delay after user sends a transaction
  - Wait for transaction to be included in the block.
  - Too little gas fees => Longer time to get included.



Sending Transaction 2

Solidity Code in Contract

```
function createRandomZombie(string _name) public {
  require(ownerZombieCount[msg.sender] == 0);
  uint randDna = _generateRandomDna(_name);
  randDna = randDna - randDna % 100;
  _createZombie(_name, randDna);
}
```

Call function in Web3.js using MetaMask

```
function createRandomZombie(name) {
  // This is going to take a while, so update the UI to let the user know
  // the transaction has been sent
  $("#txStatus").text("Creating new zombie on the blockchain. This may take
a while...");
  // Send the tx to our contract:
  return cryptoZombies.methods.createRandomZombie(name)
  .send({ from: userAccount })
  .on("receipt", function(receipt) {
    $("#txStatus").text("Successfully created " + name + "!");
    // Transaction was accepted into the blockchain, let's redraw the UI
    getZombiesByOwner(userAccount).then(displayZombies);
  })
  .on("error", function(error) {
    // Do something to alert the user their transaction has failed
    $("#txStatus").text(error);
  });
}
```

Calling Payable Function

- Function that can recieved ether.
- However, the unit in which the function received is terms of wei.

```
cryptoZombies.methods.levelUp(zombieId)
.send({ from: userAccount, value: web3js.utils.toWei("0.001", "ether") })
```

Unit	Wei Value	Wei
wei	1 wei	1
Kwei (babbage)	1e3 wei	1,000
Mwei (lovelace)	1e6 wei	1,000,000
Gwei (shannon)	1e9 wei	1,000,000,000
microether (szabo)	1e12 wei	1,000,000,000,000
milliether (finney)	1e15 wei	1,000,000,000,000,000
ether	1e18 wei	1,000,000,000,000,000,000

Test out here in the future.

https://remix.ethereum.org