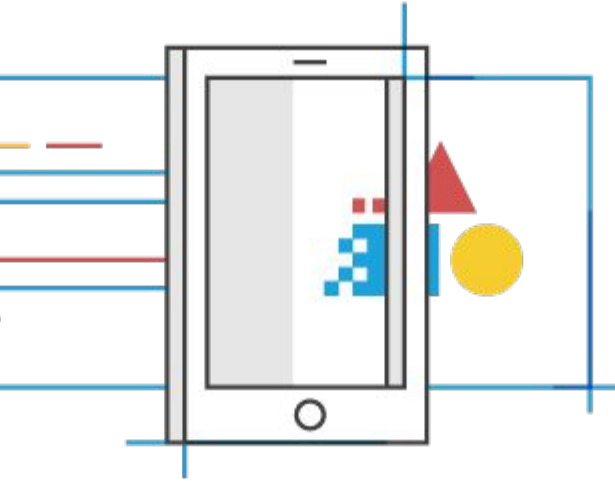




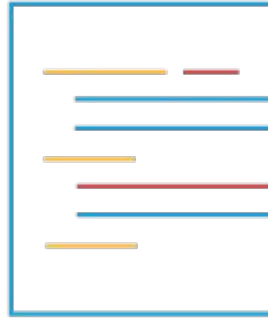
# Blockchain Labs

---

**Wifi: Lighthouse Labs, PW: 46Spadina**



# Solidity in Depth



# Overview of Today's Class

---

- **Structure**
- **Syntax**
- **Samples**

# Structure

---

# The Solidity Programming Language

- Higher level language used to create smart contracts
- Looks like JS
- Contracts as Classes (OOP/COP)
- Statically Typed
- Supports Multiple Inheritance
- Compiles to bytecode

# Structure

```
pragma solidity ^0.4.24;           // Compiler version

contract Example {                 // Contract name
    constructor(...args) {}       // Constructor

    function oneExample() {}       // method

    function() {}                  // fallback function
}
```

# Data Types

```
int a = 1;
uint256 b = 1 << 100;

bool isTrue = false;

string hello = "World";
bytes3 c = "lol";

enum MyChoices {Eat, Sleep, Repeat};

address myFriend = 0xc02c0307ab11559ca0cdcb1245439f95feafc14;

uint[3] = [1, 2, 12];
mapping(address => uint256) balances;

struct Person {
    string Name;
    uint8 age;
}
```

# Functions & their Accessors

```
function example() {};  
function example() public {}  
function example() internal {}  
function example() external {}  
function example() private {}  
function example() view {}  
function example() pure {}  
function {}
```



# Global Variables

msg	block	tx
sender	blockhash	gasprice
data	coinbase	origin
gas	difficulty	
sig	gasLimit	
value	number	
	timestamp / now	

# Address Functions

```
address myAddress = 0x3f5CE5FBFe3E9af3971dD833D26bA9b5C936f0bE;  
address myLibrary = 0x63f18e6418dc61D3B344D9Fe52810dbbB0336C35;  
  
uint256 etherBalance = myAddress.balance;  
  
myAddress.call(bytes4(keccak256("foo(uint256)")), ...args);  
myAddress.call.gas(10000).value(1 ether)("someFunction", "someArgument");  
  
myAddress.send(1 ether);  
myAddress.transfer(1 ether);  
  
myLibrary.callcode(bytes4(keccak256("myFunc(uint256)")), ...args);  
myLibrary.delegatecall(bytes4(keccak256("myFunc(uint256)")), ...args);
```

# Global Functions

```
assert(smt == true);  
require(otherthing == false);  
revert();  
  
ecrecover(hash,v,r,s);  
sha3(keccak256("DoubleHash"));  
  
addmod(x,y,k)  
mulmod(x,y,k)  
  
suicide(address myAddress);  
selfdestruct(address myAddress);
```

# Modifiers

- Used as a guard
- Runs before the function call
- Check arguments and/or valid state
- They are Inheritable properties
- Can be overwritten
- **\*\*Payable modifier\*\***

```
modifier hasEnoughFunds() {  
    require(msg.value > 1 ether);  
    _;  
}  
  
modifier argumentIsValid(uint number) {  
    require(number > 10);  
    _;  
}  
  
modifier mutex() {  
    require(lock == false);  
    lock = true;  
    _;  
    lock = false;  
}
```

# Events

- Best way to interface between contract and dapp
- Cheap storage
- Arguments will be stored in a transaction log (receipts)
- Not accessible from inside contract
- Up to 3 Indexed params
- Inheritable

```
event SomethingHappened(bytes32 arg1, uint arg2);  
emit SomethingHappened("It really did", 42);
```

```
event SomethingHappenedAndICanSearchForIt(address indexed  
canSearch, uint noSearch);
```

```
emit SomethingHappenedAndICanSearchForIt("0x..", 42);
```

# Libraries

- Keeps code DRY
- Saves on gas
- Cannot be inherited
- Cannot receive Ether
- “For” syntax

```
library myNewLib {  
  
    function life(bytes8 otherArg) returns(uint8) {  
        return 42;  
    }  
  
    function isTheMeaning(bytes8 arg) returns (bytes8){  
        return "of";  
    }  
}  
  
contract myContract {  
    using myNewLib for bytes8;  
    bytes8 what = "what";  
    uint8 answer = what.isTheMeaning().life();  
}
```



# Inheritance

- Solidity Supports multiple inheritance
- Overloading is allowed
- Only 1 contract on blockchain
- “Super” calls parent class function
- Multiple inheritance

```
contract Life {
    uint searchFor;
    function Life(uint answer) {
        searchFor = answer;
    }


    function meaningOfLife() returns(uint) {
        return searchFor;
    }
}

contract Human is Life(42) {
    event Print(uint ans);
    bool isHuman = true;
    function meaningOfLife() returns(uint) {
        if (isHuman == true) {
            return super.meaningOfLife();
        }
        return 0;
    }
}
```

# Creating a Contract Inside a Contract







```
contract Baby {  
    function resetName(string newName);  
}  
  
contract Human {  
    Baby deployedBaby = Baby(0x0)  
}
```

```
contract Baby {  
    string name;  
    function constructor(string givenName) {  
        name = givenName;  
    }  
    function resetName(string newName) { name = newName; }  
}  
  
contract Human {  
    Baby[] public babies;  
    address[] public babyAddresses;  
    function createBaby(string name) {  
        Baby myNewBaby = new Baby(name);  
        babies.push(myNewBaby);  
        babyAddresses.push(address(myNewBaby));  
    }  
    function lookAtBaby(uint index) public view returns (Baby) {  
        return babies[index];  
    }  
    function lookAtBabyAddress(uint index) public view returns(address) {  
        return babyAddresses[index];  
    }  
}
```