GO WITH THE FLOW

FIRST EDITION – Chapter 2 REV 1

Kevin Thomas & Jacob Tucker

Forward

Blockchain economies are the future of how we approach intellectual property. Some will lecture that this is a phase and will crash and burn as the general public will not adapt it.

This thinking reminds me of a time when I ran a 300 baud, dial-up BBS on my Commodore 64 and some would say computers are a phase and will never catch on. There too clunky and there is entirely too much information to learn to make them practical. They will never last through the 80s.

Time obviously showed a different result.

Go With The Flow is a FREE e-book based on the teaching of Jacob Tucker who is a Smart Contract Developer/DApp Engineer at Decentology.

Jacob is at the bleeding-edge of this technology and has created a FREE video tutorial series to take you from zero to hero in modern Smart Contract Development.

Each chapter will reflect each video in addition to providing the source code so you could more easily follow along.

I would suggest you first watch each video and then to reinforce what you just watched, read each chapter in this book and manually work through all of the code examples. Like any athlete they must practice. Coding is no different.

You can find the video series here on YouTube at https://www.youtube.com/watch?v=iVevnipJbHo&list=PLvcQxi9WyGdF32YuZABVTx-t3-FsBNCN2.

Let's dive in!

Table Of Contents

Chapter 1: What is the Flow Blockchain and Cadence?

Flow is a fast, decentralized, and developer-friendly blockchain, designed as the foundation for a new generation of games, apps, and the digital assets that power them.

Flow was originally designed to support a blockchain game called CryptoKitties where you purchase, collect, breed and sell virtual cats.

An NFT, non-fungible token, is a unique and non-interchangeable unit of data stored on a digital ledger. NFTs can be associated with easily-reproducible items such as photos, videos, audio, and other types of digital files as unique items, and use blockchain technology to give the NFT a public proof of ownership.

So we ask ourselves, why Flow? The traditional smart contract language of Solidity unfortunately allows you to too easily shoot yourself in the foot as it is easy to make mistakes. For example, in 2016, an overlooked vulnerability in an Ethereum DAO smart contract (Decentralized Autonomous Organization) saw millions of dollars siphoned from a smart contract, eventually leading to a fork in Ethereum and two separate active blockchains (Ethereum and Ethereum Classic).

Flow is focused on security and safety and focused on a better Developer experience.

The next logical question is what is Flow? Flow is a blockchain and Cadence is the smart contract programming lanauge. Smart contracts are digital contracts stored on a blockchain that are automatically executed when predetermined terms and conditions are met.

Cadence is focused on five unique pillars which are Safety and Security, Clarity, Approachability, Developer Experience and Intuiting Ownership with Resources as more details can be found at https://docs.onflow.org/cadence.

In a resource-oriented language like Cadence, resources directly tie an asset to the account that owns it by saving the resource in the account’s storage. As a result, ownership isn’t centralized in a single, central smart contract’s storage. Instead, each account owns and stores its own assets, and the assets can be transferred freely between accounts without the need for arbitration by a central smart contract. Every resource has exactly one owner. Resources cannot go

out of scope. If a function or transaction removes a resource from an account’s storage, it either needs to end the transaction in an account's storage, or it needs to be explicitly and safely deleted. There is no “garbage collection” for resources.

Resources change how assets are used in a programming environment to better resemble assets in the real world. Users store their currencies and assets in their own account, in their own wallet storage, and they can do with them as they wish. Users can define custom logic and structures for resources that give them flexibility with how they are stored. Additionally, because everyone stores their own assets, the calculation and charging of state rent is fair and balanced across all users in the network.

In our next chapter we will begin our introduction to smart contracts.

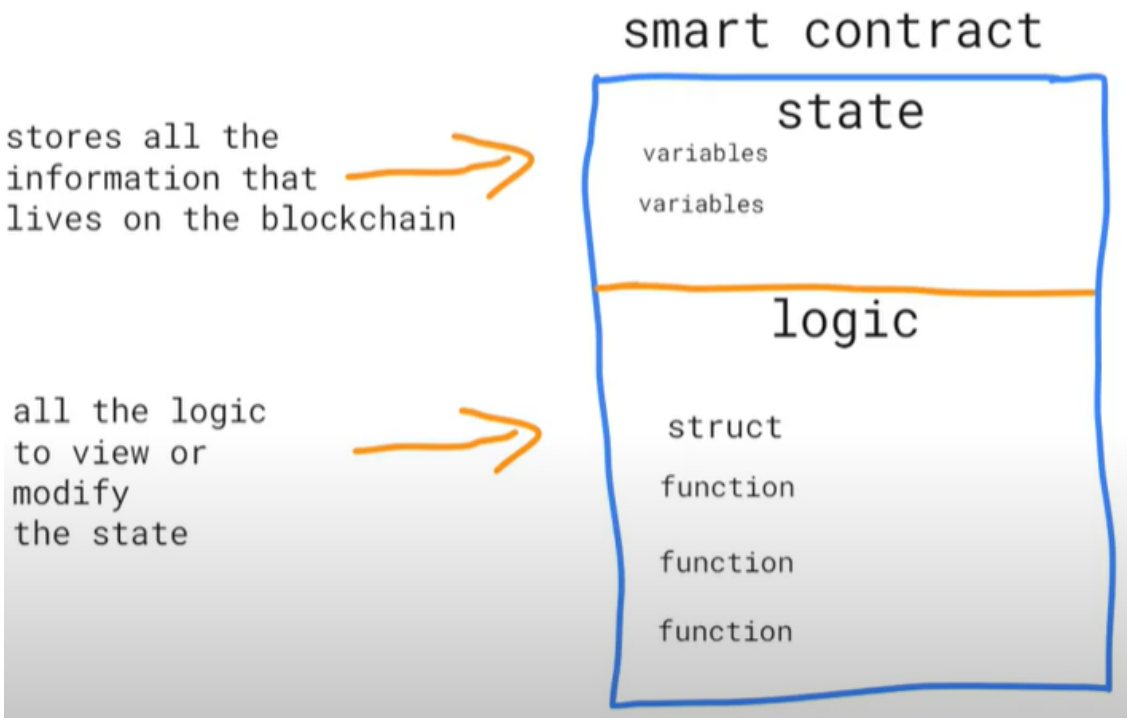
Chapter 2: Intro to Smart Contracts

In this chapter we will get familiar with smart contracts. In chapter 1 we defined smart contracts as digital contracts stored on a blockchain that are automatically executed when predetermined terms and conditions are met. A smart contract is a program that verifies and executes the performance of a contract without the need for a trusted third party.

A Smart Contract is broken down into two categories, state and logic. State will hold all of the information that will live on the blockchain. You could, for example, set a name variable to Jacob and it will forever live on that blockchain. State contains variables such as dictionaries, strings, integers, etc.

The logic of a smart contract are the actual rules governing the smart contract such as modifying and viewing the state. You could, for example change the name variable of Jacob to Tucker. Logic contains structures and functions, etc.

Let's refer to the below diagram.



Let’s visit the Flow Playground to dive into our first Smart Contract by visiting https://play.onflow.org.

Unlike a real smart contract that will sit out there on the blockchain, this is a local file that we can experiment with and develop out.

We will delete everything out of the template and strip it down to a more readable solution.

```
pub contract HelloWorld {
    //
    // state
    //
    pub let greeting: String

    //
    // logic
    //
    pub fun hello(): String {
        return self.greeting
    }

    init() {
        self.greeting = "Hello, World!"
    }
}
```

The code breakdown is as follows. The contract is called *HelloWorld* which is indicated on the first line.

We then have our state, our variable called *greeting* which is a string.

Finally we have our logic which views or modifies our state. Our logic begins with a function, *hello()* which returns a string which in our case is our greeting.

If we called *hello()* on the blockchain it would return us our *greeting* variable.

The *init()* function simply populates the variable with a string literal, in our case, "*Hello, World!*"

Let's click the deploy button which simulates sending a smart contract to the actual Flow Blockchain mainnet.

```
Deployment > [1] > Deployed Contract To: 0x01
```

The above line is the output from the deployed smart contract. *0x01* is an address. On the Flow Blockchain, smart contracts are put into an address (account). Every account on the Flow Blockchain has an

address associated with it therefore we put this contract to this very account of *0x01*.

Let's click on the Script tab on the left hand side under SCRIPT TEMPLATES which will reveal our script code.

```
import HelloWorld from 0x01

pub fun main() {
  log(HelloWorld.hello())
}
```

In our local Flow Blockchain there is a smart contract called *HelloWorld* that is in account or address *0x01* and it currently has a state called *greeting* that contains the value, *“Hello, World!”*

We are simply importing the *HelloWorld* contract from address *0x01* and using the built-in log function to call *hello()* which will return our string.

Let's click execute and review the result below.

```
Script > [1] "Hello, World!"
Script > [2] Result > {"type":"Void"}
```

We see our string as expected and because the function did not return any value we see the result as type void which is expected.

All a script does is read the blockchain, it is free, it does not require any gas or currency.

In our next chapter we will learn about transactions and scripts.