Algorand Tutorials

MPhil FinTech 2024 Coding Tutorials

Francisco Rutayebesibwa
rtyfra001@myuct.ac.za
February 2024

Slide credits:
Julian Kanjare, MPhil FIntech 2021 Coding Tutorials
A/Prof Co-Pierre Georg: How Entrepreneurs in Emerging Markets can master the Blockchain Technology
Cosimo Bassi: Algorand Deep Dive UZH Summer School 2021

Gary Lu: https://www.blocksauce.io/ & Finhub Developer Workshops

Agenda

- Introductions
- Crash course on the blockchain
- Introduction to Algorand
- Interacting with the Algorand Blockchain
- Smart Contracts
- PyTEAL

TUTORIAL GOALS & FORMAT

GOAL

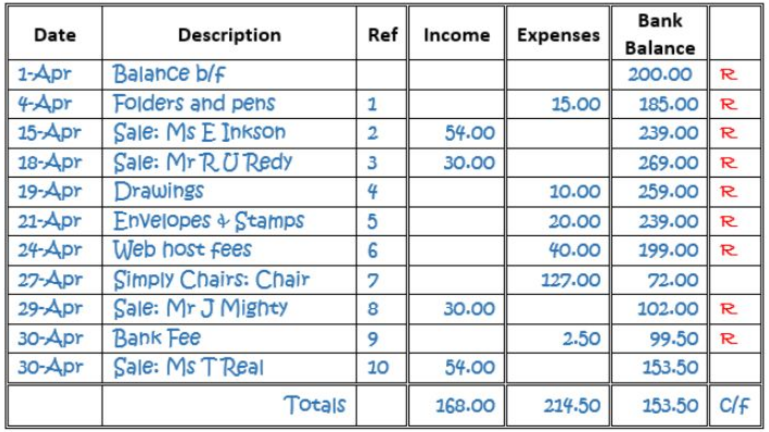
- Understand enough theory to get you up and going in the realm of Blockchain development, specifically in the Algorand Ecosystem.
- Understand the principles of Algorand blockchain through practical examples.

FORMAT

- Each session will consist of a theory component and practical component.
- These slides do not cover all the functionalities provided in the Algorand ecosystem, hence do not use these slides alone.

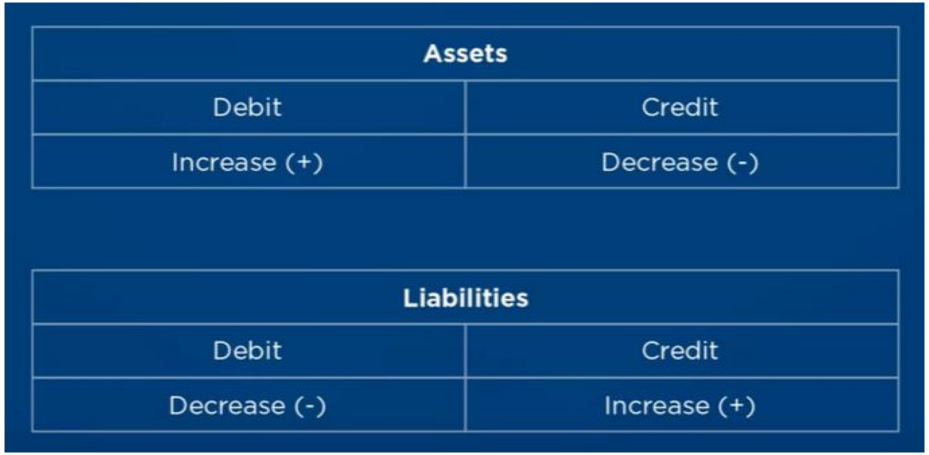
Blockchain Crash Course

Single Entry & Double Entry Bookkeeping



Date	Description	Ref	Income	Expenses	Bank Balance	
1-Apr	Balance b/f				200.00	R
4-Apr	Folders and pens	1		15.00	185.00	R
15-Apr	Sale: Ms E Inkson	2	54.00		239.00	R
18-Apr	Sale: Mr R U Redy	3	30.00		269.00	R
19-Apr	Drawings	4		10.00	259.00	R
21-Apr	Envelopes & Stamps	5		20.00	239.00	R
24-Apr	Web host fees	6		40.00	199.00	R
27-Apr	Simply Chairs: Chair	7		127.00	72.00	
29-Apr	Sale: Mr J Mighty	8	30.00		102.00	R
30-Apr	Bank Fee	9		2.50	99.50	R
30-Apr	Sale: Ms T Real	10	54.00		153.50	
	Totals		168.00	214.50	153.50	C/F

Single Entry Bookkeeping



Assets	
Debit	Credit
Increase (+)	Decrease (-)

Liabilities	
Debit	Credit
Decrease (-)	Increase (+)

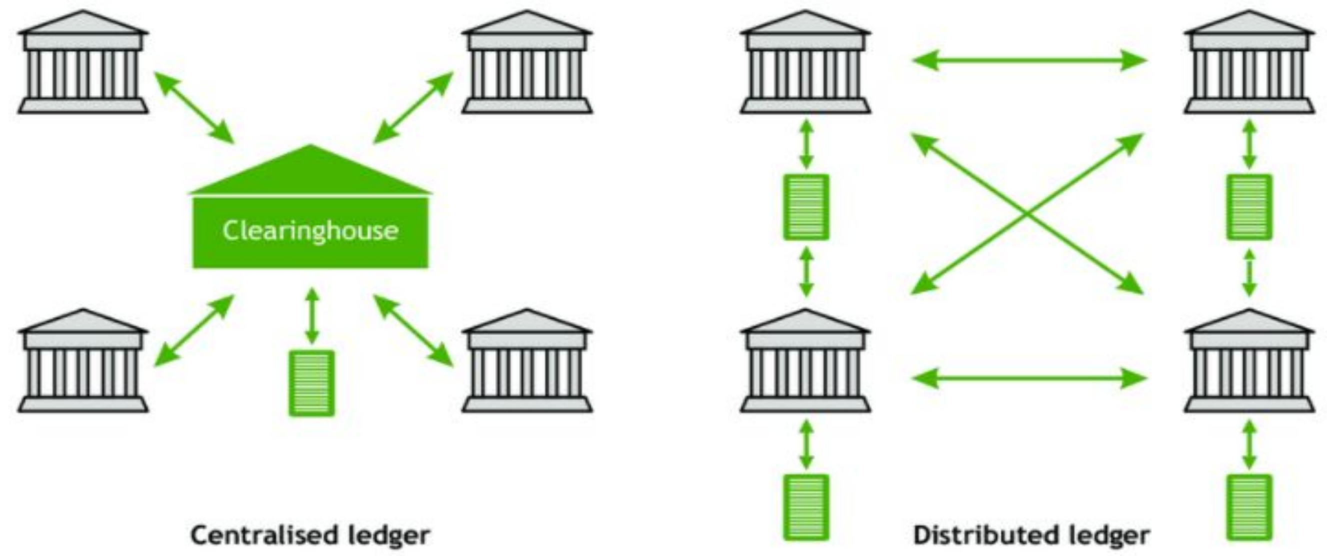
Double Entry Bookkeeping

Blockchain: Definition

"The blockchain is a distributed database that is operated on a peer-to-peer network and can be used to record transactions, ownership, and securely store any type of information in blocks."

Co-Pierre Georg, How entrepreneurs in emerging countries can master the Blockchain technology

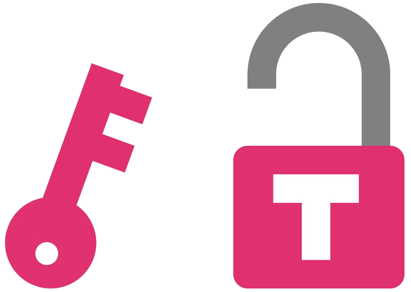
Centeralised ledger vs Distributed ledger



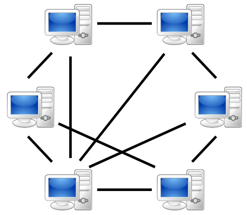
Traditional centralised ledger and a distributed ledger

KEY COMPONENTS OF A BLOCKCHAIN

Distributed Ledger

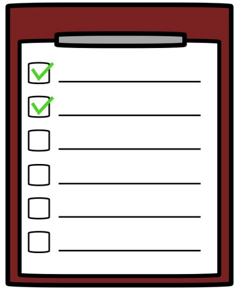


Cryptography



Peer-2-Peer Network

Consensus Mechanism



Validity Rules

Terminology:

- *Blockchain*: A data structure used to create a decentralized or distributed ledger, where the ledger is an **append-only** list of records.
- *Block*: Dataset containing multiple records and cryptographically secured elements to be appended onto the blockchain.

Blockchain Platforms

CARDANO

bitcoin

Stellar

ethereum

ripple

HYPERLEDGER FABRIC

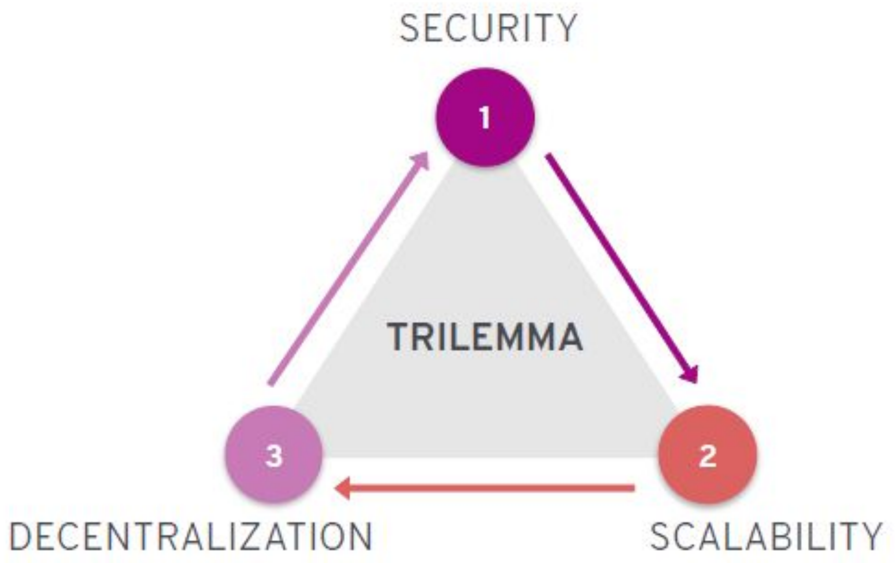
Algorand

Algorand

Blockchain Trilema, consensus mechanism (PPoS), Layer I Features, & Algorand Sandbox

Discussion - Why another blockchain? What are the differences, shortcomings etc? How fast is the space moving?

The Blockchain Trilema



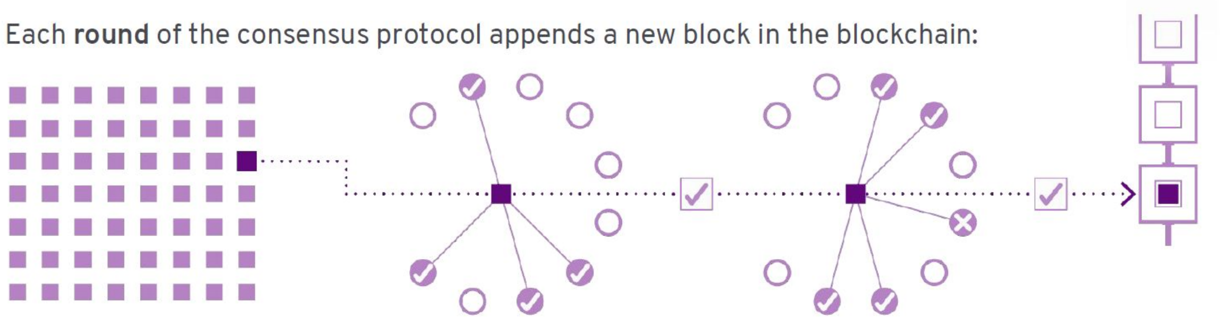
Consensus Mechanisms

	Description	Drawbacks
Proof of Work	Miners compete against each other to solve a cryptographic puzzle. The winner gets to append the next block and earns a reward for their effort.	● Huge electrical consumption ● Concentration of governance in few mining farms. ● Soft-forking of the blockchain
Bonded Proof of Stake	Validators bind their stake, to show their commitment in validating and appending a new block. Any validators caught misbehaving are punished.	● Risk of economic barriers
Delegated Proof of Stake	Users delegate the validation process to a fixed committee, through weighted votings based on their stakes.	● Governance is centralized. ● Known delegate nodes, hence exposed to DDos attacks.

Pure Proof of State Mechanism (PPoS)

Pure Proof of Stake, in short

Each **round** of the consensus protocol appends a new block in the blockchain:



An account is elected to **propose** the **next block**	A **committee** is elected to **filter** and **vote** on the **block proposals**	A **new committee** is elected to reach a quorum and **certify** the **block**	The **new block** is **appended** to the **blockchain**

Through the **cryptographic lottery**, an **online account** is elected with probability directly proportional to its stake: **each ALGO corresponds to an attempt** to win the lottery!

Algorand PPoS Consensus

Scalable	billions of users
Efficient	1000 TPS (10x work in progress)
Fast	< 5s per block
Low fees	0.001 ALGO per txn
No Soft Forks	prob. $< 10^{-18}$

- **Instant Transaction Finality**
- **Minimal hardware node requirements**
- **No delegation or binding of the stake**
- **No minimum stake**
- **Carbon negative**
- **Secure with respect DDoS**
- **Network Partitioning resilience**

Algorand Layer-1 Features I

- Algorand Standard Assets
    - Provides a standardized, Layer-1 mechanism to represent any type of asset on the Algorand blockchain
- Atomic Transactions
    - Provide a trustless solution in Layer-1.
    - Atomic Transfers offer a secure way to simultaneously transfer a number of assets among a number of parties.
    - Upto 16 transactions can be ‘bundled’ together i.e. all or nothing.
    - No need of third party like an escrow service
    -
- Rekeying
    - This enables an Algorand account holder to maintain a static public address while dynamically rotating the authoritative private spending key(s)

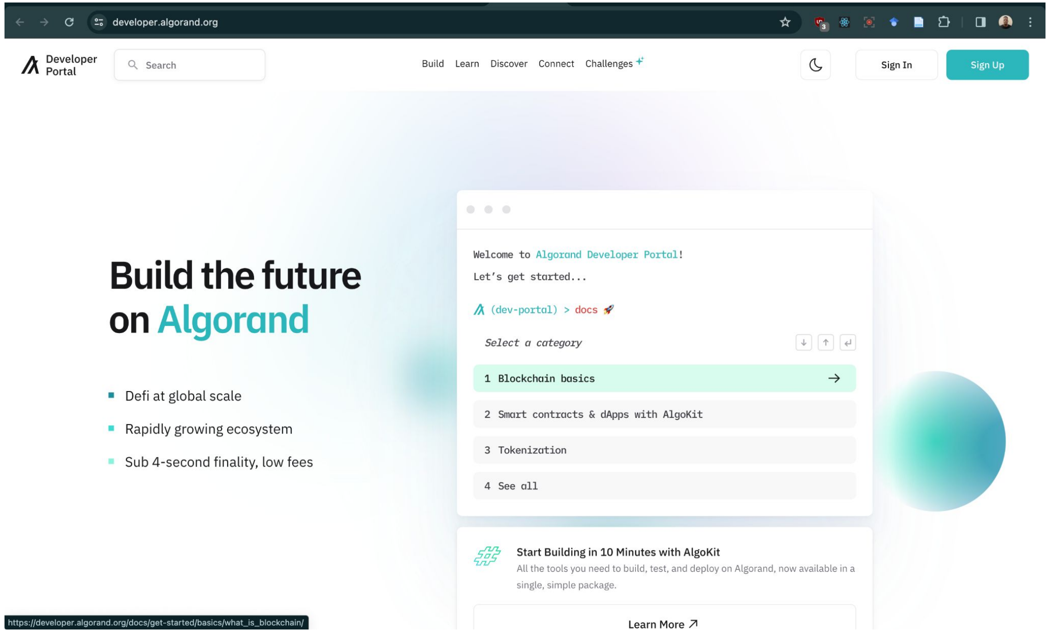
Algorand Layer-1 Features II

- Smart Contracts are deterministic programs through which complex decentralized trustless applications can be executed on the Algorand Virtual Machine (AVM).
- AVM runs on every node in the Algorand blockchain. This virtual machine contains a stack engine that evaluates smart contracts and smart signatures against the transactions they're called with.
- Smart contracts, also referred to as stateful smart contracts, contain logic that is deployed and can be remotely called from any node on the Algorand blockchain.
- Smart signatures, also referred to as stateless contracts, contain logic that is used to sign transactions, commonly for signature delegation. The logic of the smart signature is submitted with the transaction

Algorand

Ecosystem

Algorand Ecosystem I


developer.algorand.org

Developer Portal

Search

Build Learn Discover Connect Challenges

Sign In Sign Up

Build the future on Algorand

- Defi at global scale
- Rapidly growing ecosystem
- Sub 4-second finality, low fees

```
Welcome to Algorand Developer Portal!
Let's get started...

(dev-portal) > docs 🚀

Select a category

1 Blockchain basics
2 Smart contracts & dApps with AlgoKit
3 Tokenization
4 See all
```

Start Building in 10 Minutes with AlgoKit

All the tools you need to build, test, and deploy on Algorand, now available in a single, simple package.

Learn More ↗

https://developer.algorand.org/docs/get-started/basics/what_is_blockchain/

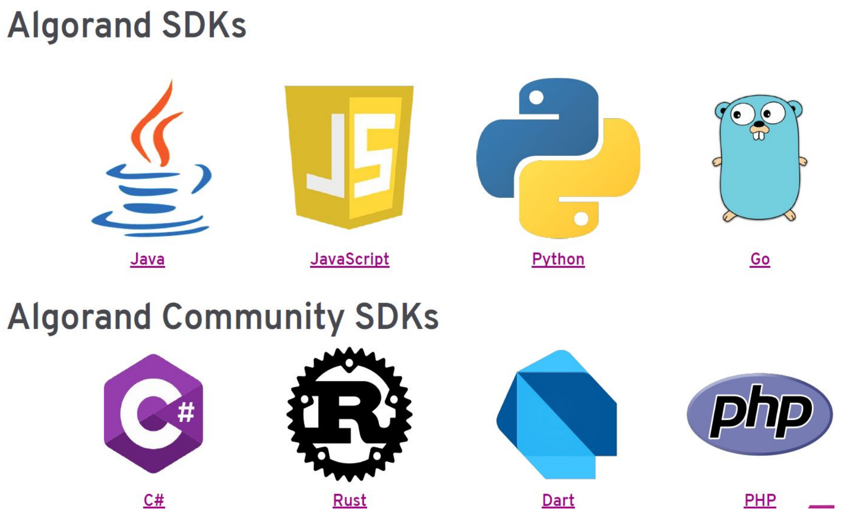
https://developer.algorand.org/

Algorand Ecosystem II

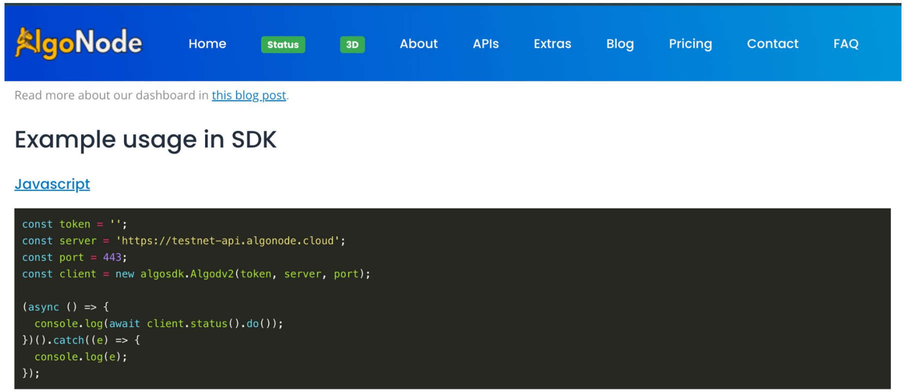
- Networks
  - MainNet: where Algo and real assets are traded (i.e. production)
  - TestNet: Mirror of MainNet (i.e., current protocol features)
  - BetaNet: Bleeding edge protocol features
- Since the TestNet and BetaNet are independent networks and separate from the Algorand MainNet, they require their own ‘test’ Algo tokens
- Test tokens are distributed via faucets i.e. web-based services that provide free tokens to users of a testnet, enabling them to experiment with the blockchain network features without spending real tokens on the mainnet
- Algorand Faucets (a.k.a dispensers)
  - TestNet: https://bank.betanet.algodev.network
  - BetaNet: https://bank.betanet.algodev.network

Algorand SDKs

- A software development kit (SDK) is a set of tools provided by the manufacturer of a hardware platform, operating system (OS), or programming language.
- SDKs help software developers create applications for that specific platform or programming language.
- https://developer.algorand.org/docs/sdks/



Interacting with the Algorand Blockchain



Read more about our dashboard in this blog post.

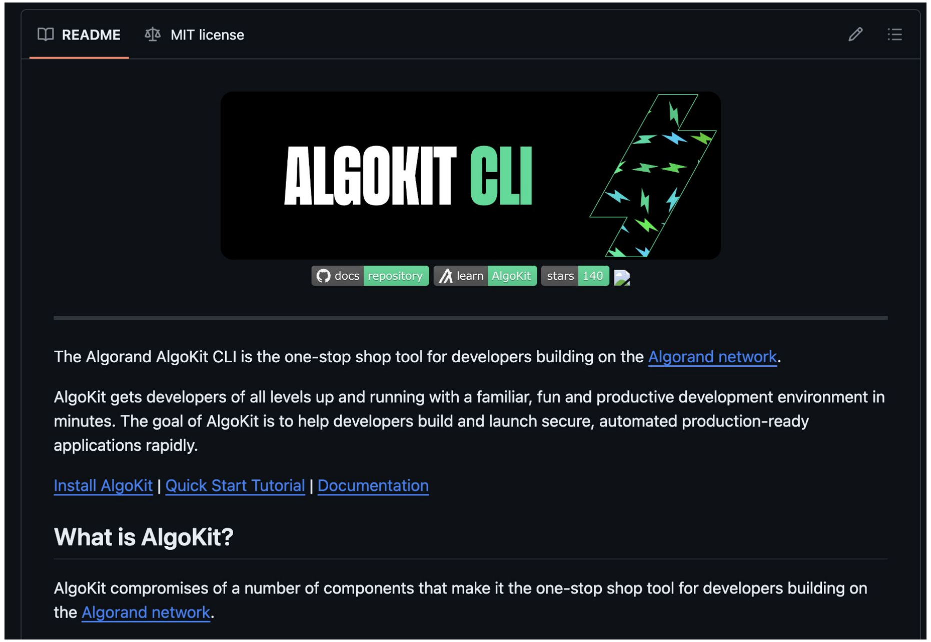
Example usage in SDK

Javascript

```
const token = '';
const server = 'https://testnet-api.algonode.cloud';
const port = 443;
const client = new algosdk.Algodv2(token, server, port);

(async () => {
  console.log(await client.status().do());
})().catch((e) => {
  console.log(e);
});
```

https://nodely.io/api/#example-usage-in-sdk



The Algorand AlgoKit CLI is the one-stop shop tool for developers building on the Algorand network.

AlgoKit gets developers of all levels up and running with a familiar, fun and productive development environment in minutes. The goal of AlgoKit is to help developers build and launch secure, automated production-ready applications rapidly.

Install AlgoKit | Quick Start Tutorial | Documentation

What is AlgoKit?

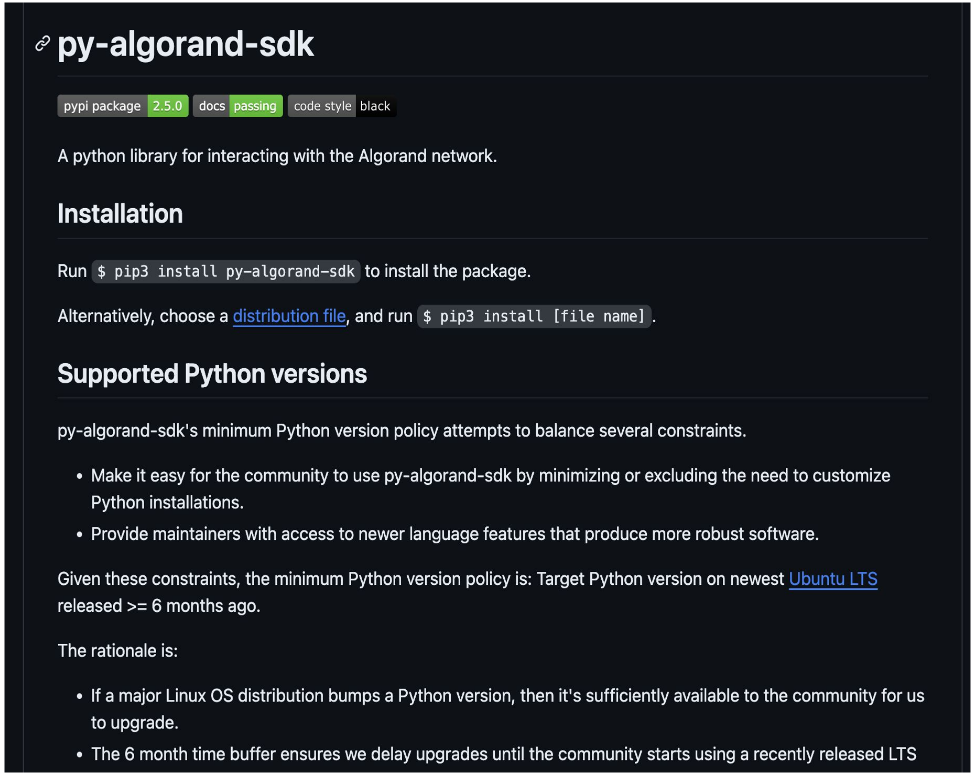
AlgoKit compromises of a number of components that make it the one-stop shop tool for developers building on the Algorand network.

https://github.com/algorandfoundation/algokit-cli

Algorand

Python SDK

Interacting with the Algorand Network



py-algorand-sdk

pypi package 2.5.0 docs passing code style black

A python library for interacting with the Algorand network.

Installation

Run `$ pip3 install py-algorand-sdk` to install the package.

Alternatively, choose a distribution file, and run `$ pip3 install [file name]`.

Supported Python versions

py-algorand-sdk's minimum Python version policy attempts to balance several constraints.

- Make it easy for the community to use py-algorand-sdk by minimizing or excluding the need to customize Python installations.
- Provide maintainers with access to newer language features that produce more robust software.

Given these constraints, the minimum Python version policy is: Target Python version on newest Ubuntu LTS released >= 6 months ago.

The rationale is:

- If a major Linux OS distribution bumps a Python version, then it's sufficiently available to the community for us to upgrade.
- The 6 month time buffer ensures we delay upgrades until the community starts using a recently released LTS

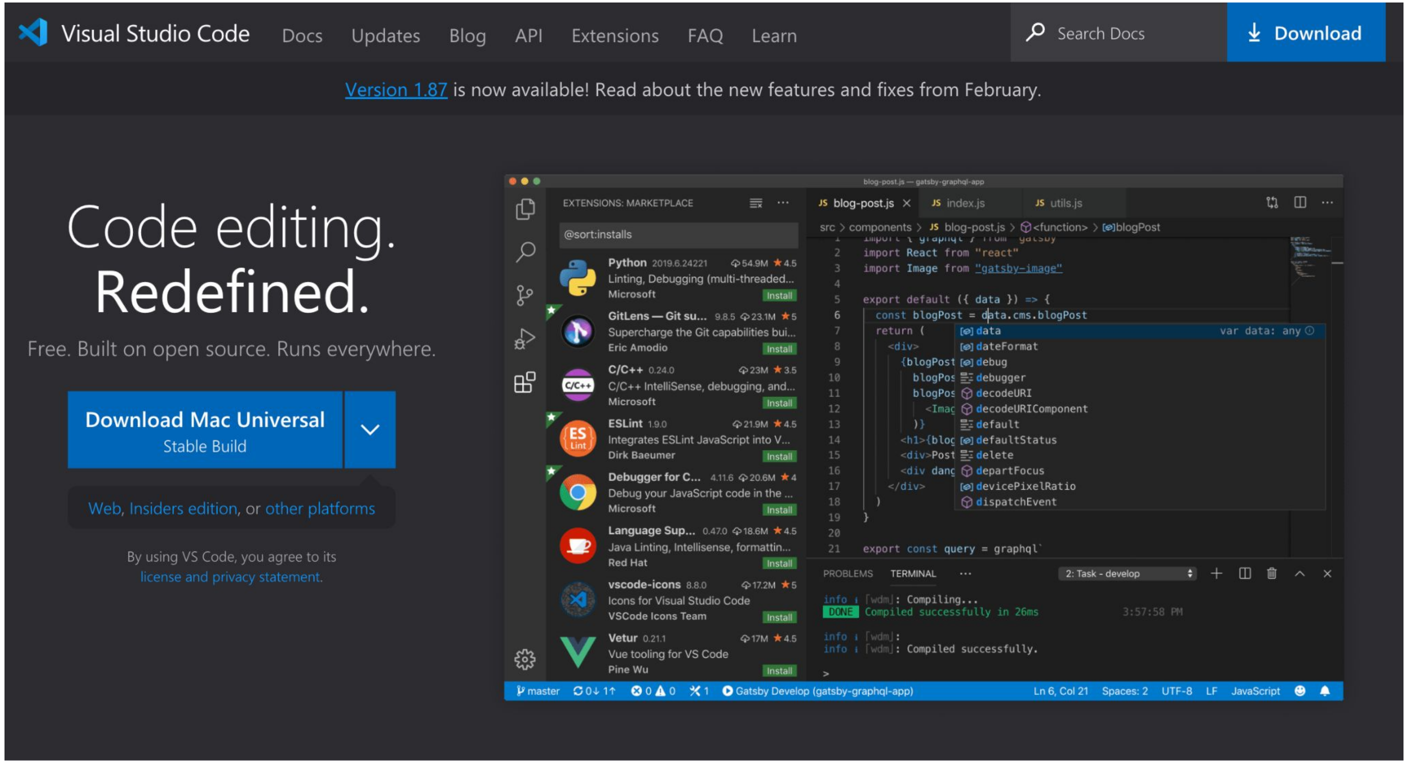
https://github.com/algorand/py-algorand-sdk

Your first transaction using Python SDK

Prac Time!

- Create a folder/directory where you will store all python scripts for the fintech tutorial.
- Follow the prerequisite instructions on the following link: Getting started with py-algorand
- Once you have completed the prerequisite steps, copy/download the python script: generate_account.py and try running the script.
  - Make sure to store the: Public key/Address, Private key, and mnemonic as you'll use it in future tutorials.
- Use https://bank.testnet.algorand.network/ to fund your address.
- If you have completed the above, explore the transaction_api_boilerplate.py script. Do not worry if it does not make sense, as we will go through it in depth in the next tutorial.
  - The following link describes most of the code: Your First transaction using Python SDK

Visual Studio Code


Visual Studio Code   Docs   Updates   Blog   API   Extensions   FAQ   Learn   Search Docs   Download

Version 1.87 is now available! Read about the new features and fixes from February.

Code editing.
Redefined.

Free. Built on open source. Runs everywhere.

Download Mac Universal
Stable Build

Web, Insiders edition, or other platforms

By using VS Code, you agree to its
license and privacy statement.

blog-post.js — gatsby-graphql-app

EXTENSIONS: MARKETPLACE

@sort:installs

Python 2019.6.24221 54.9M ★4.5
Linting, Debugging (multi-threaded...
Microsoft Install

GitLens — Git su... 9.8.5 23.1M ★5
Supercharge the Git capabilities bui...
Eric Amodio Install

C/C++ 0.24.0 23M ★3.5
C/C++ IntelliSense, debugging, and...
Microsoft Install

ESLint 1.9.0 21.9M ★4.5
Integrates ESLint JavaScript into V...
Dirk Baeumer Install

Debugger for C... 4.11.6 20.6M ★4
Debug your JavaScript code in the ...
Microsoft Install

Language Sup... 0.47.0 18.6M ★4.5
Java Linting, Intellisense, formattin...
Red Hat Install

vscode-icons 8.8.0 17.2M ★5
Icons for Visual Studio Code
VSCode Icons Team Install

Vetur 0.21.1 17M ★4.5
Vue tooling for VS Code
Pine Wu Install

blog-post.js   index.js   utils.js

src > components > blog-post.js > <function> > blogPost

```
import { graphql } from "gatsby"
import React from "react"
import Image from "gatsby-image"

export default ({ data }) => {
  const blogPost = data.cms.blogPost
  return (        data                     var data: any
    <div>         dateFormat
      {blogPost   debug
        blogPos   debugger
        blogPos   decodeURI
          <Imag   decodeURIComponent
        )}        default
      <h1>{blog   defaultStatus
      <div>Post   delete
      <div dang   departFocus
    </div>        devicePixelRatio
  )               dispatchEvent
}

export const query = graphql`
```

PROBLEMS   TERMINAL   2: Task - develop

```
info ｉ ｢wdm｣: Compiling...
DONE  Compiled successfully in 26ms                    3:57:58 PM

info ｉ ｢wdm｣:
info ｉ ｢wdm｣: Compiled successfully.

>
```

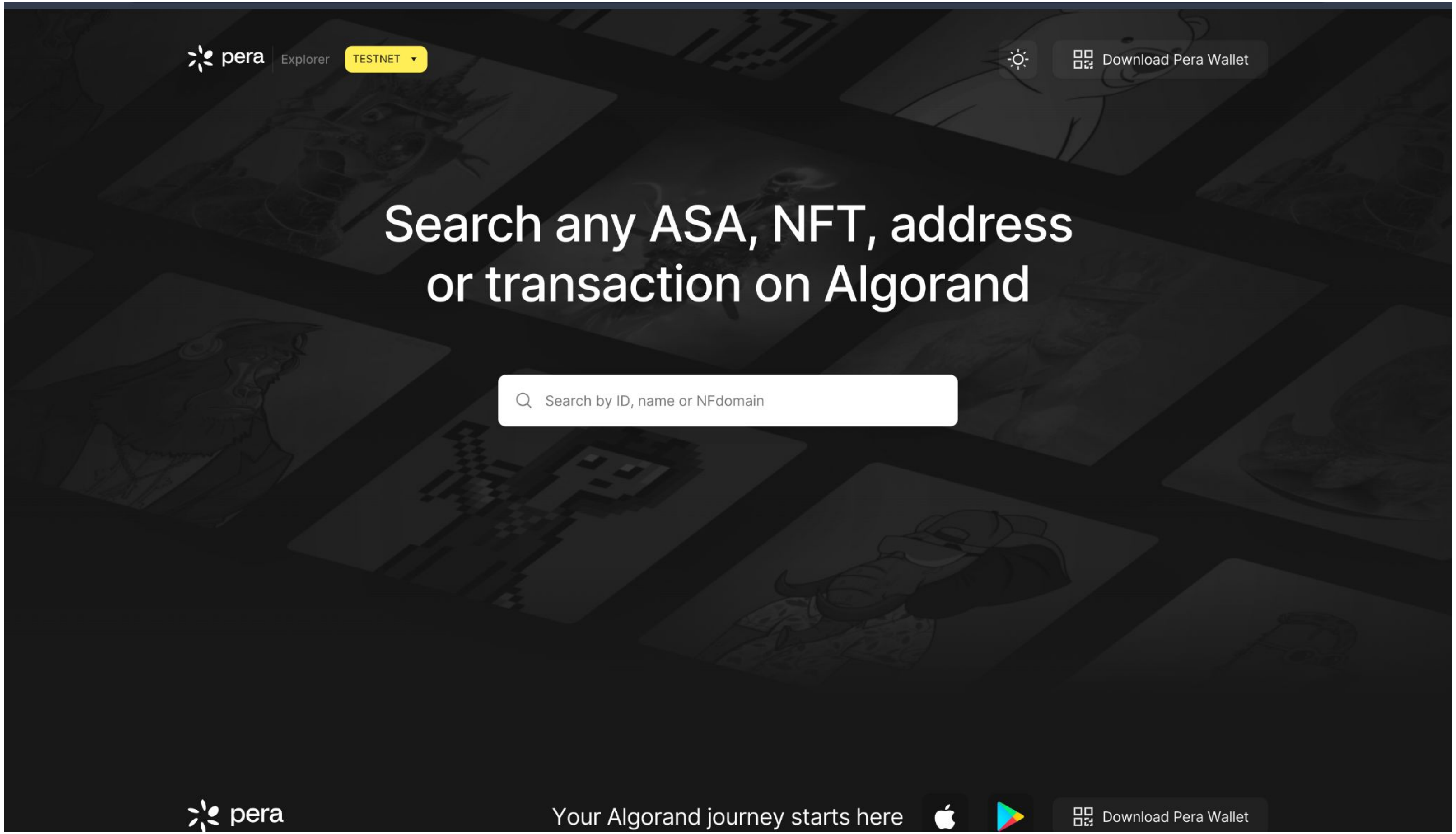
master   0↓ 1↑   0   0   1   Gatsby Develop (gatsby-graphql-app)   Ln 6, Col 21   Spaces: 2   UTF-8   LF   JavaScript

IntelliSense   Run and Debug   Built-in Git   Extensions

https://code.visualstudio.com/

Algorand

Pera Wallet Blockchain Explorer


pera
Explorer
TESTNET
Download Pera Wallet

Search any ASA, NFT, address or transaction on Algorand

Search by ID, name or NFdomain
pera
Your Algorand journey starts here
Download Pera Wallet

https://testnet.explorer.perawallet.app/

Algorand

Sending ALGOs to different accounts

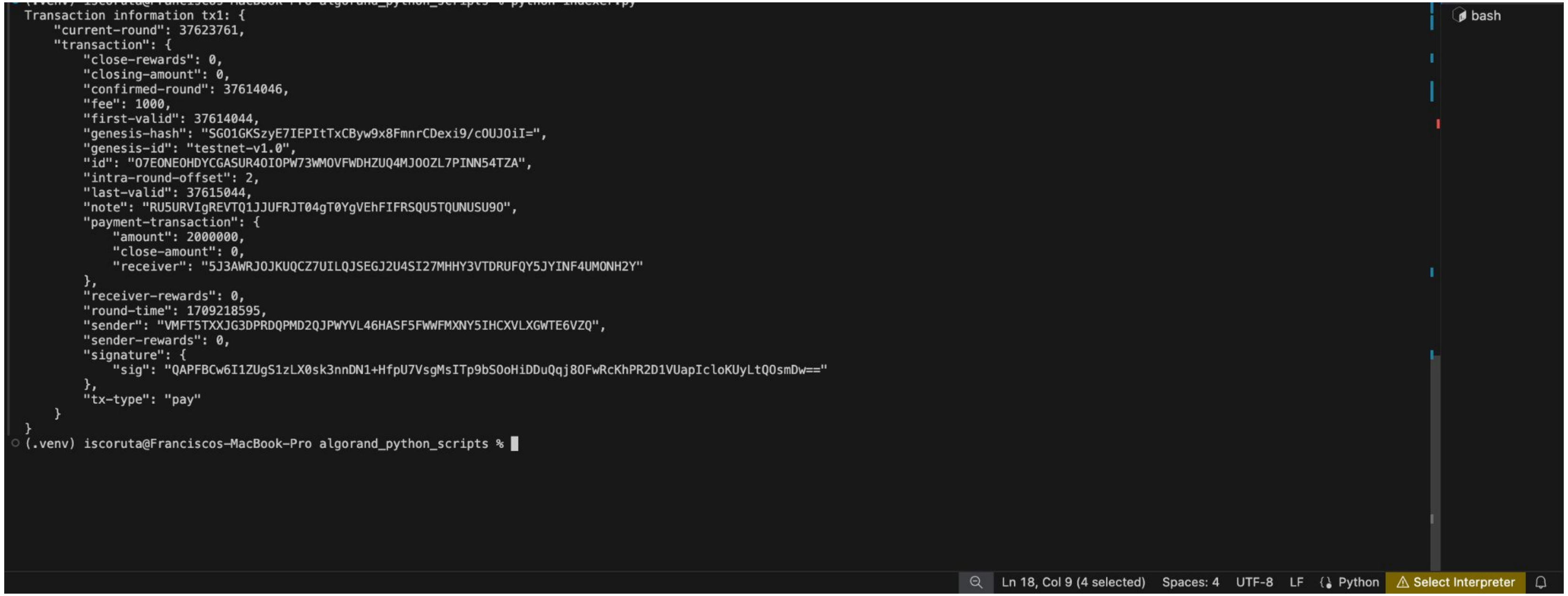
Prac Time: Sending ALGOs to a class member

- **Objective:** The aim of this tutorial is to send ALGOs to a class member. Make sure you have their Algorand Public address.
- Next, use the ***transaction_api.py*** script as a starting point.
  - In the directory you are working in, make sure you have a python virtual environment setup & activated beforehand.
  - Manipulate the python code to work for your own address details.
    - If you do not have an address, use the ***generate_account.py*** to generate an Algorand address.
    - Use https://bank.testnet.algorand.network/ to fund your address.
  - You can check transactions that have happened on your address on Pera Algo Explorer Testnet.

Algorand

Algorand Indexer

What is “wrong” with the below picture?



```
Transaction information tx1: {
    "current-round": 37623761,
    "transaction": {
        "close-rewards": 0,
        "closing-amount": 0,
        "confirmed-round": 37614046,
        "fee": 1000,
        "first-valid": 37614044,
        "genesis-hash": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
        "genesis-id": "testnet-v1.0",
        "id": "O7EONEOHDYCGASUR4OIOPW73WMOVFWDHZUQ4MJOOZL7PINN54TZA",
        "intra-round-offset": 2,
        "last-valid": 37615044,
        "note": "RU5URVIgREVTQ1JJUFRJT04gT0YgVEhFIFRSQU5TQUNUSU9O",
        "payment-transaction": {
            "amount": 2000000,
            "close-amount": 0,
            "receiver": "5J3AWRJOJKUQCZ7UILQJSEGJ2U4SI27MHHY3VTDRUFQY5JYINF4UMONH2Y"
        },
        "receiver-rewards": 0,
        "round-time": 1709218595,
        "sender": "VMFT5TXXJG3DPRDQPMD2QJPWYVL46HASF5FWWFMXNY5IHCXVLXGWTE6VZQ",
        "sender-rewards": 0,
        "signature": {
            "sig": "QAPFBCw6I1ZUgS1zLX0sk3nnDN1+HfpU7VsgMsITp9bSOoHiDDuQqj8OFwRcKhPR2D1VUapIcloKUyLtQOsmDw=="
        },
        "tx-type": "pay"
    }
}
(.venv) iscoruta@Franciscos-MacBook-Pro algorand_python_scripts %
```

https://developer.algorand.org/docs/get-details/indexer/

Algorand

Algorand Standard Assets (ASAs)

Algorand Standard Assets (ASAs)

- ASAs = official name for assets on Algorand blockchain network.
- ASAs can represent stablecoins, loyalty points, cryptocurrencies, etc.
- ASAs can represent single, unique assets like a deed for a house, collectable items, etc.
- For every asset an account creates or owns, its minimum balance is increased by 0.1 Algos (100,000 microAlgos).
- This minimum balance requirement will be placed on the original creator as long as the asset has not been destroyed. Transferring the asset does not alleviate the creator's minimum balance requirement.
- Before a new asset can be transferred to a specific account the receiver must opt-in to receive the asset.
- If any transaction is issued that would violate the minimum balance requirements, the transaction will fail.

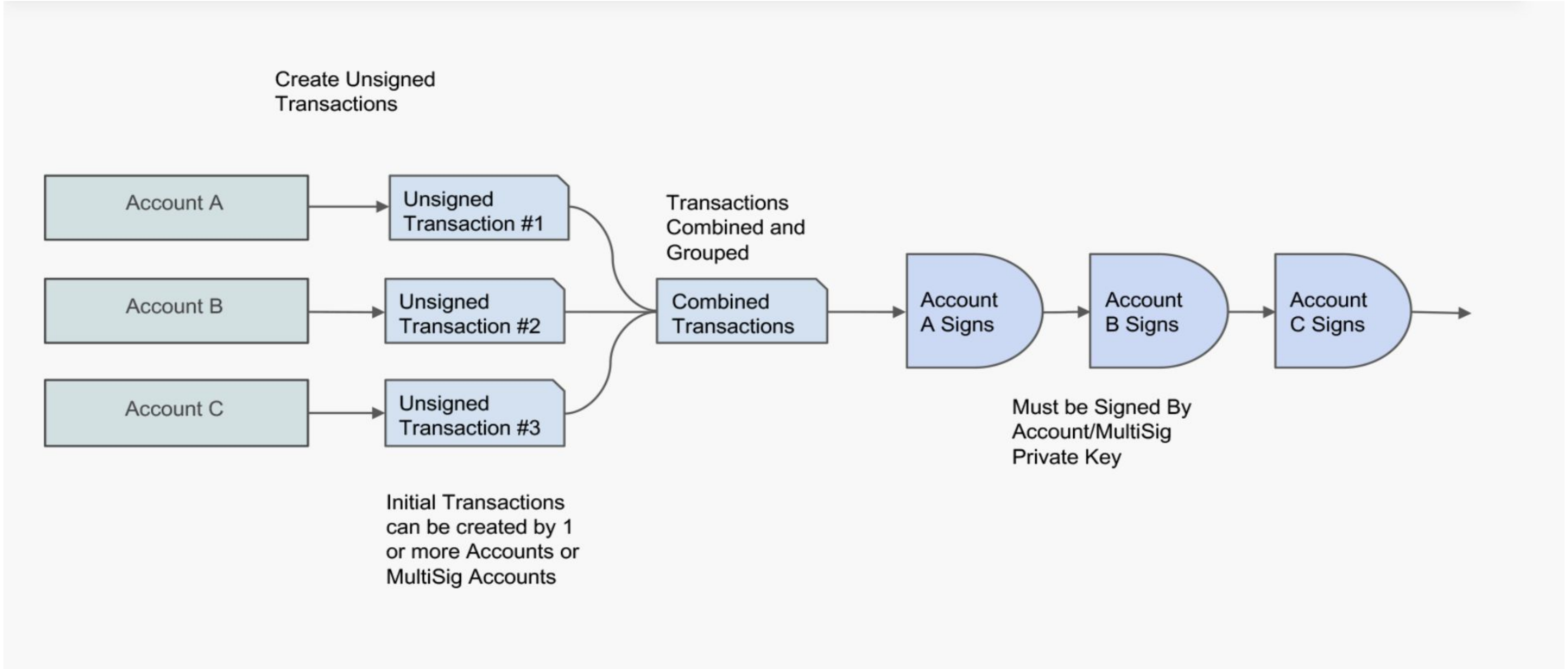
Algorand

Atomic Transfers/Group Transactions

Atomic Transactions I

- Asset swaps have been the bread and butter of trade. I have an asset that you want, and you have an asset (e.g., money) that I want
- On Algorand, asset swaps, whether involving two or more parties, can be executed in an *atomic* (i.e., indivisible) manner.
- All assets involved are swapped *simultaneously* in the *same* transaction group.
- This approach provides at least two crucial advantages:
    - *It eradicates counterparty risk.*
    - *It eradicates any possible "midstream" mishaps*

Atomic Transactions II


Create Unsigned Transactions
Account A
Account B
Account C
Unsigned Transaction #1
Unsigned Transaction #2
Unsigned Transaction #3
Transactions Combined and Grouped
Combined Transactions
Account A Signs
Account B Signs
Account C Signs
Must be Signed By Account/MultiSig Private Key
Initial Transactions can be created by 1 or more Accounts or MultiSig Accounts

Atomic Transfer Flow

https://developer.algorand.org/docs/get-details/atomic_transfers/

Prac Time: Mint Digital Currency

- **Objective:** The aim of this practical is to create digital Currency (of your choice) on the Algorand Testnet, and distribute it to another account (it can be your own or a class member's address).
- Use the ***create_nft.py*** script as a starting point, however make sure you play around with the different settings when creating digital currency (e.g. explore having different addresses with different access roles, what happens when you set defaultFreeze true or false)
  - In the directory you are working in, make sure you have a python virtual environment setup & activated beforehand.
  - Manipulate the python code to work for your scenario.
    - If you do not have an address, use the ***generate_account.py*** to generate an Algorand address.
    - Use https://bank.testnet.algorand.network/ to fund your address.
  - You can check transactions that have happened on Algo Explorer Testnet.

Prac Time: Atomic transactions

- **Objective:** The aim of this practical is to prepare a group transaction between two accounts. One account should be sending ALGOs whilst the other account should be sending the digital currency you minted in the earlier prac.
- Use the ***atomic_transactions.py*** script as a starting point.
  - In the directory you are working in, make sure you have a python virtual environment setup & activated beforehand.
  - Manipulate the python code to work for your scenario (ask for help if your running into errors).
    - If you do not have an address, use the ***generate_account.py*** to generate an Algorand address.
    - Use https://bank.testnet.algorand.network/ to fund your address.
  - You can check transactions that have happened on Algo Explorer Testnet.

Algorand

Rekey

Algorand Rekeying

- Rekeying is a powerful protocol feature which enables an Algorand account holder to maintain a static public address while dynamically rotating the authoritative private spending key(s).
- This is accomplished by issuing a "rekey-to transaction" which sets the *authorized address* field within the account object.
- Future transaction authorization using the account's public address must be provided by the spending key(s) associated with the *authorized address* which may be a single key address, MultiSig address or LogicSig program address.

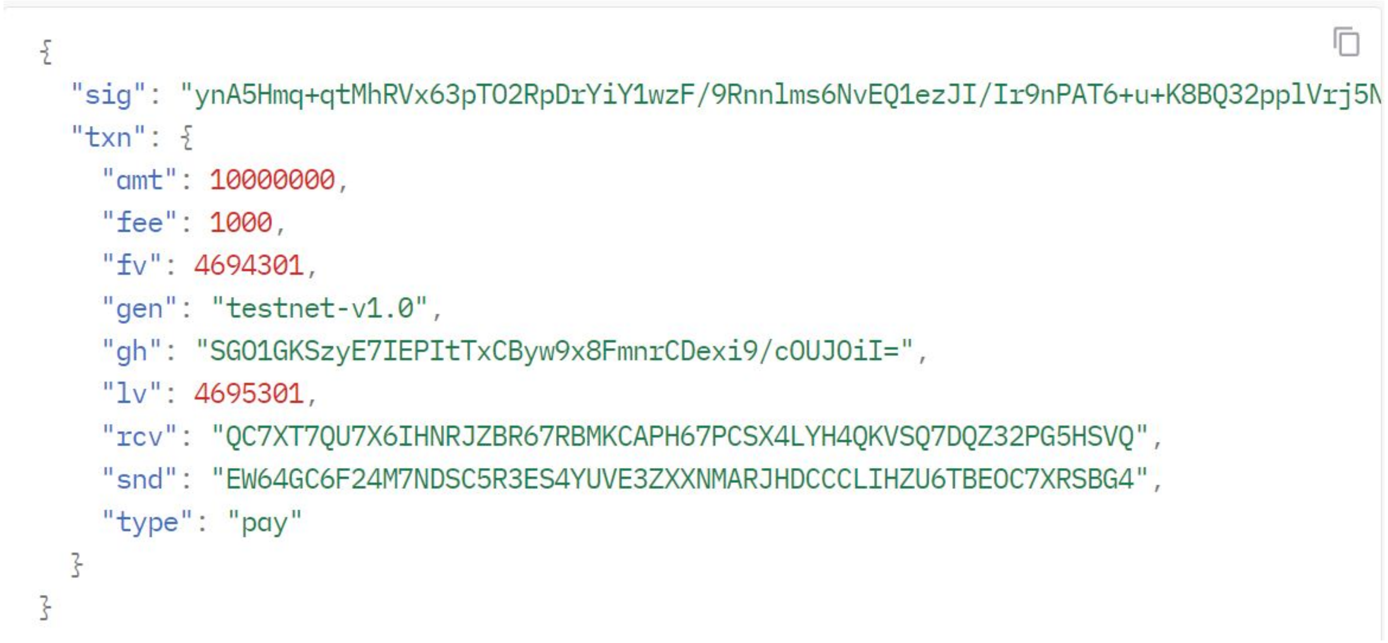
Algorand

Signatures.

Transaction Authorization

- Transactions must be authorized before being sent to the network
- Authorization is done through the addition of a signature to the transaction object
- A Signed Transaction object includes the transaction and a type of signature
- Three types of signatures are available:
    - Single Signatures
    - Multi Signatures
    - Logic Signatures

Single Signatures



```
{
  "sig": "ynA5Hmq+qtMhRVx63pTO2RpDrYiY1wzF/9Rnnlms6NvEQ1ezJI/Ir9nPAT6+u+K8BQ32pplVrj5N
  "txn": {
    "amt": 10000000,
    "fee": 1000,
    "fv": 4694301,
    "gen": "testnet-v1.0",
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 4695301,
    "rcv": "QC7XT7QU7X6IHNRJZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HSVQ",
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
    "type": "pay"
  }
}
```

[Link to Single Signatures Documentation]

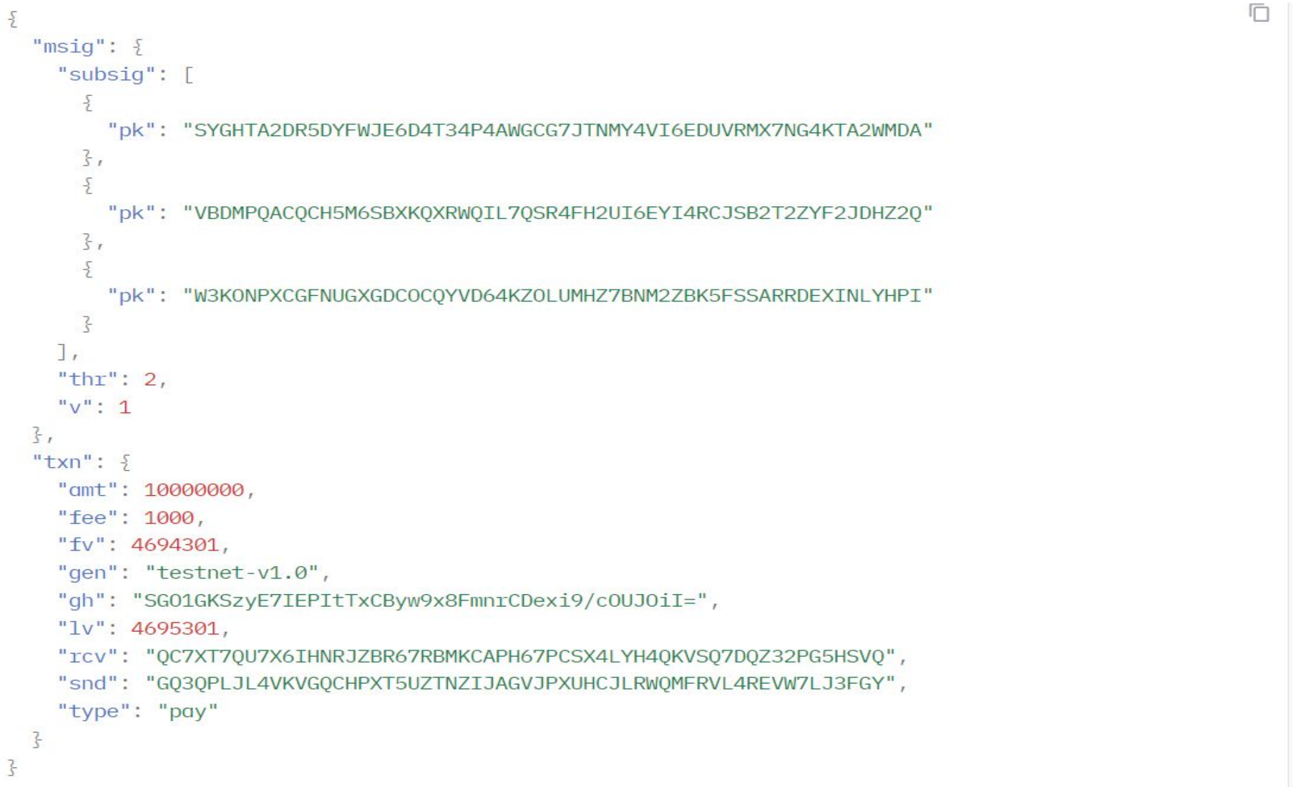
Multisignature Accounts

- Multisignature accounts require a subset of signatures, equal to or greater than the threshold value, from associated private keys to authorize a transaction
- Multisignature accounts are a logical representation of an ordered set of addresses with a threshold and version
- The threshold determines how many signatures are required to process any transaction from this multisignature account
- Creating a multisignature account with the same addresses but in a different order produces a different address

Why use Multisignature Accounts

- Reasons to use Multisignature Accounts:
  - Multisignature accounts can provide an extra layer of security on an account by requiring multiple signatures to authorize spending
  - They can be used to create cryptographically secure governance structures for an account, where spending is authorized by a subset of multiple users' keys
- Reasons **not** to use Multisignature accounts
  - Multisignature accounts trade off convenience for security, and may be overly complex for scenarios where security or governance is not critical.

Multisignature Account Examples



```
{
  "msig": {
    "subsig": [
      {
        "pk": "SYGHTA2DR5DYFWJE6D4T34P4AWGCG7JTNMY4VI6EDUVRMX7NG4KTA2WMDA"
      },
      {
        "pk": "VBDMPQACQCH5M6SBXKQXRWQIL7QSR4FH2UI6EYI4RCJSB2T2ZYF2JDHZ2Q"
      },
      {
        "pk": "W3KONPXCGFNUGXGDCOCQYVD64KZOLUMHZ7BNM2ZBK5FSSARRDEXINLYHPI"
      }
    ],
    "thr": 2,
    "v": 1
  },
  "txn": {
    "amt": 10000000,
    "fee": 1000,
    "fv": 4694301,
    "gen": "testnet-v1.0",
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 4695301,
    "rcv": "QC7XT7QU7X6IHNRJZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HSVQ",
    "snd": "GQ3QPLJL4VKVGQCHPXT5UZTNZIJAGVJPXUHCJLRWQMFRVL4REVW7LJ3FGY",
    "type": "pay"
  }
}
```

Prac Time: Rekeying & Multisignature Account

- **Objective:** The aim of this practical is to go through two python scripts that look at the multisignature & rekeying functionalities Algorand Python SDK kit does. Try to understand what is happening. If there are functions/classes you do not understand, hover on the function to get definitions of what the function does, and what arguments are used.

Algorand

Smart Contracts

Algorand Smart Contracts

- Algorand Smart Contracts (ASC1) operate on layer-1 and serve various functions on the blockchain.
- Two main categories of smart contracts: stateful and stateless contracts (also known as smart signatures).
- Both types of contracts are written in the Transaction Execution Approval Language (TEAL), an assembly-like language interpreted by the Algorand Virtual Machine (AVM) within an Algorand node.
- TEAL programs can be written manually or by using the PyTEAL compiler in Python.

Smart (Stateful) Contracts

- Once deployed, smart contracts are remotely callable from any node in the Algorand blockchain.
- Once deployed, the on-chain instantiation of the contract is referred to as an Application and assigned an Application Id.
- triggered by a specific type of transaction called an Application Call transaction
- Stateful contracts are used to manage complex on-chain data and application states.
- These contracts keep track of the state of the blockchain and can be used for various applications such as voting, auctions, and decentralized finance.
- A stateful contract is evaluated at every transaction, and the state is updated accordingly.
- TEAL programs for stateful contracts include global variables, which are used to store and manage the state of the contract.

Signature (Stateless) Contracts

- Stateless contracts are used for simple, one-off transactions.
- These contracts do not manage any state and are used for tasks such as atomic swaps and multisig transactions.
- A stateless contract is evaluated only once, during the initial transaction.
- TEAL programs for stateless contracts do not use global variables, as there is no state to manage.

Transaction Execution Application Language (TEAL)

- TEAL is an assembly-like language used for writing Algorand smart contracts.
- TEAL is a stack-based language, with operations such as add, subtract, multiply, and divide.
- TEAL also includes conditional statements, loops, and jumps, allowing for complex program logic.
- TEAL programs are compiled to bytecode and executed by the Algorand Virtual Machine (AVM) within an Algorand node.

Algorand

PyTeal

What is PyTEAL

- They are Algorand Smart contracts with Python like syntax.
- Collection PyTeal expressions.
- PyTEAL provides a higher-level abstraction of TEAL, making it easier to write and understand smart contracts.
- PyTEAL code is compiled to TEAL byte code, which can be deployed to the Algorand blockchain.
- PyTEAL also includes a testing framework, allowing for easy testing and debugging of smart contracts.

Prac Time: Smart contract on Algorand blockchain

- **Objective:** This practical is different from previous practicals. In software development, you are faced with technology you have not seen before. There is a python file called voting.py that can be found on Vula, or Github. The aim of the practical is to understand what the smart contract is doing. Once you have a sense of what is going on, try uploading the smart contract. If you are unable to do this, do not worry; as we will go through it in the next tutorial session.
- Make use of the following resources:
  - PyTeal for Beginners (Full Course) | Python-like Algorand Smart Contract Language Tutorial - YouTube
  - Introduction to Smart Contracts - Algorand Developer Portal
  - Smart Contract Overview - Algorand Developer Portal
  - Interact with smart contracts

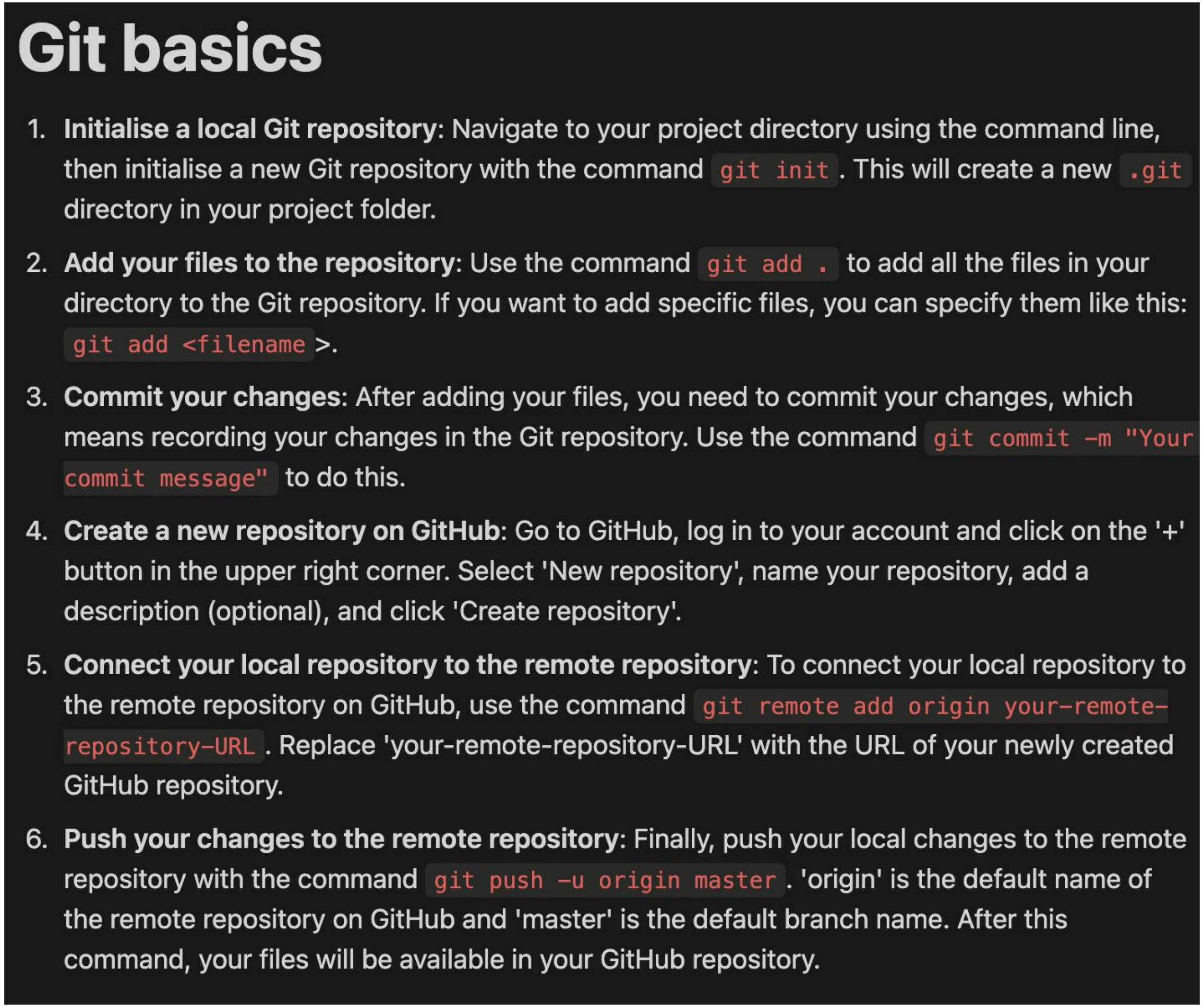
Version Control

How do you manage different versions/variations of your project?

Git & Github

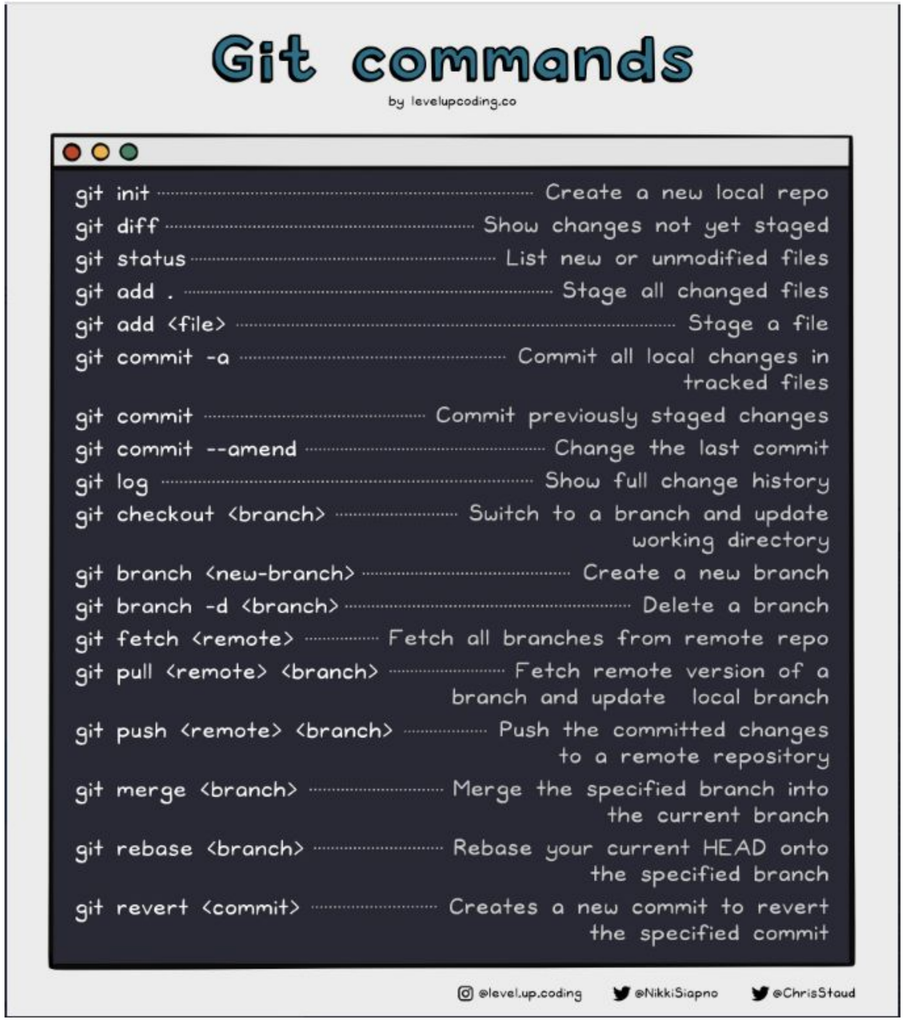
- Git is a version control software.
- Git is used as a command line tool and ran locally.
- Git allows us to track various versions/branches of a codebase.
    - In addition it allows us to synchronous different versions of the same codebase (i.e. interact with local and remote branches).

- Github is a web application that allows git repositories to be hosted remotely.
- Provides extra functionality on top of git
- Mainly used by organisations to manage projects.



Git basics

1. **Initialise a local Git repository**: Navigate to your project directory using the command line, then initialise a new Git repository with the command `git init`. This will create a new `.git` directory in your project folder.
2. **Add your files to the repository**: Use the command `git add .` to add all the files in your directory to the Git repository. If you want to add specific files, you can specify them like this: `git add <filename`>.
3. **Commit your changes**: After adding your files, you need to commit your changes, which means recording your changes in the Git repository. Use the command `git commit -m "Your commit message"` to do this.
4. **Create a new repository on GitHub**: Go to GitHub, log in to your account and click on the '+' button in the upper right corner. Select 'New repository', name your repository, add a description (optional), and click 'Create repository'.
5. **Connect your local repository to the remote repository**: To connect your local repository to the remote repository on GitHub, use the command `git remote add origin your-remote-repository-URL`. Replace 'your-remote-repository-URL' with the URL of your newly created GitHub repository.
6. **Push your changes to the remote repository**: Finally, push your local changes to the remote repository with the command `git push -u origin master`. 'origin' is the default name of the remote repository on GitHub and 'master' is the default branch name. After this command, your files will be available in your GitHub repository.



Git commands

by levelupcoding.co

git init	Create a new local repo
git diff	Show changes not yet staged
git status	List new or unmodified files
git add .	Stage all changed files
git add <file>	Stage a file
git commit -a	Commit all local changes in tracked files
git commit	Commit previously staged changes
git commit --amend	Change the last commit
git log	Show full change history
git checkout <branch>	Switch to a branch and update working directory
git branch <new-branch>	Create a new branch
git branch -d <branch>	Delete a branch
git fetch <remote>	Fetch all branches from remote repo
git pull <remote> <branch>	Fetch remote version of a branch and update local branch
git push <remote> <branch>	Push the committed changes to a remote repository
git merge <branch>	Merge the specified branch into the current branch
git rebase <branch>	Rebase your current HEAD onto the specified branch
git revert <commit>	Creates a new commit to revert the specified commit

@level.up.coding @NikkiSiapno @ChrisStaud

Prac Time: Initializing a git repository.

- If you have not done this yet, download git (and Git Bash if you are using a Windows OS)
- The aim of this task is to:
  - Initialise a local git repository of the work you have done so far.
  - Create a git repository on Github, and link it to your
  - Next, push the local repository to your remote repository.

References & Resources

- Algorand Python Scripts Github Repo
- Solving the "Blockchain Trilemma" | Algorand Technologies
- SMARTER THAN SMART CONTRACTS | Algorand Technologies
- Your First Transaction using the Python SDK
- Algorand Standard Assets (ASAs)
- Working with ASA using Python | Algorand Developer Portal
- Atomic transfers - Algorand Developer Portal
- Signatures - Algorand Developer Portal
- Rekeying - Algorand Developer Portal
- Introduction to Smart Contract - Algorand Developer Portal
- PyTeal for Beginners (Full Course) | Python-like Algorand Smart Contract Language Tutorial - YouTube