



# Launch Your Algorand Developer Environment



#Decipher2022

# Ryan R. Fox

Developer Advocate  
@Algorand



how it started...



...how it's going

#Decipher2022



@ryanRfox



# Ryan R. Fox

Developer Advocate

@Algorand

# Let's Connect

#Decipher2022



# #BUILDL

#Decipher2022

# #BUIDL

## Smart Contracts

#Decipher2022

# #BUIDL

# dApps

#Decipher2022

# #BUIDL

## DeFi

#Decipher2022

# #BUILDL

## Games

#Decipher2022

# #BUIDL

## Smart NFTs

#Decipher2022

# #BUIDL

## Randomness

#Decipher2022

# #BUIDL

## Oracles

#Decipher2022

# #BUIDL

## Bridges

#Decipher2022

# #BUIDL

# Community

#Decipher2022

# #BUIDL

# Web3 Future

#Decipher2022

#BUIDL

Algorand

#Decipher2022

#AlgoFam

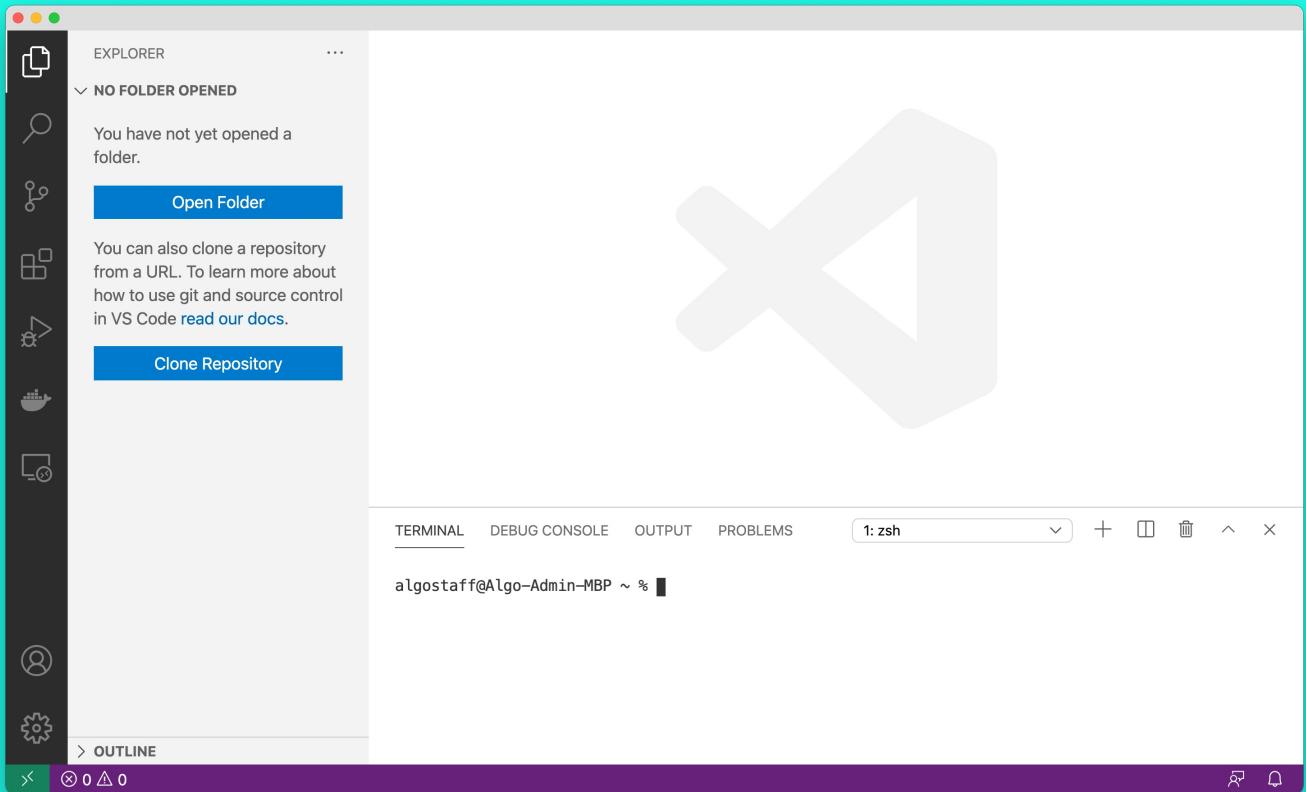
Algorand

#Decipher2022

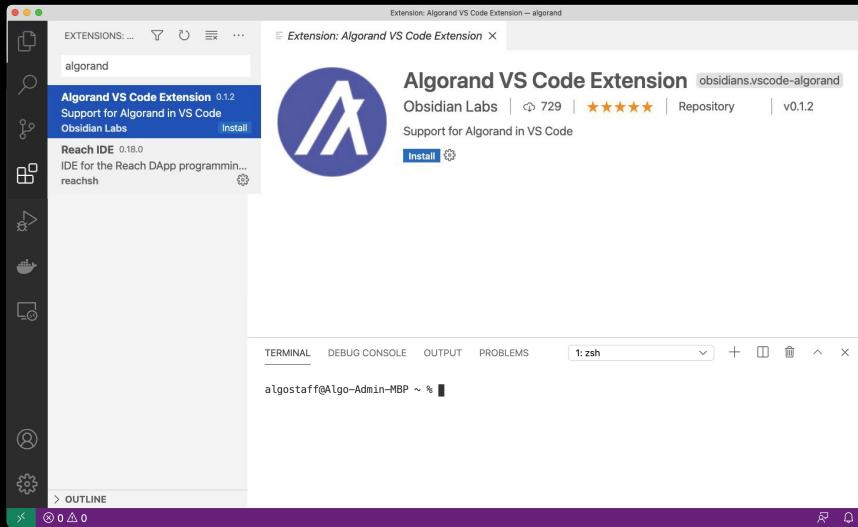
@AlgoDevs  
Algorand

#Decipher2022

# Developer Environment Setup



# Recommendation: VSCode



<https://code.visualstudio.com/Download>



#Decipher2022

# Developer Environment Components

Infrastructure



SDK



Language + Framework



\$ whoami

pc

mac



# macOS prerequisites

Docker Desktop     $\geq 4.8.0$   
Python                 $\geq 3.10$



# Windows prerequisites

WSL2

Docker Desktop  $\geq 4.8.0$

Python

$\geq 3.10$



# Windows System for Linux (WSL2)



<https://learn.microsoft.com/en-us/windows/wsl/install>



#Decipher2022

# WSL2 Installation

```
> wsl --install  
Installing: Virtual Machine Platform  
Virtual Machine Platform has been installed.  
Installing: Windows Subsystem for Linux  
Windows Subsystem for Linux has been installed.  
Installing: Ubuntu  
Ubuntu has been installed.  
The requested operation is successful.
```



# WSL2 Configuration

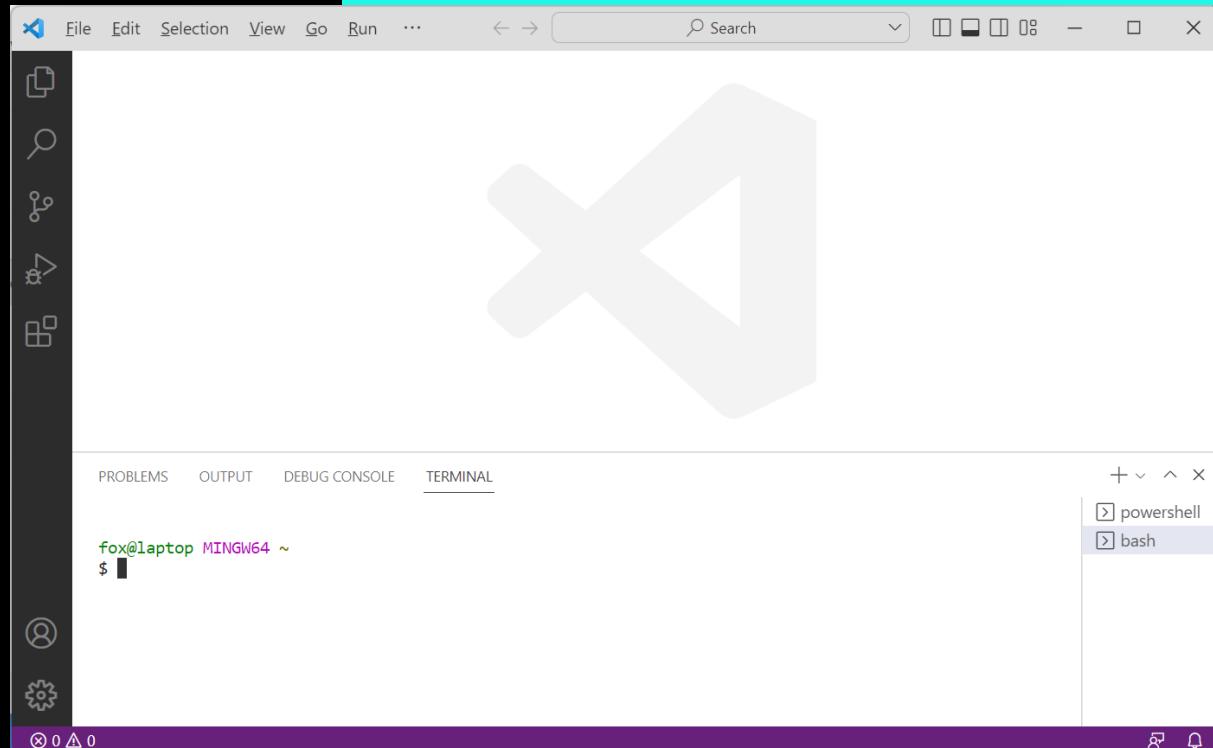
1. Reboot Windows
2. Start Ubuntu
3. Prompted to add **username and password**

```
username@hostname:~$
```

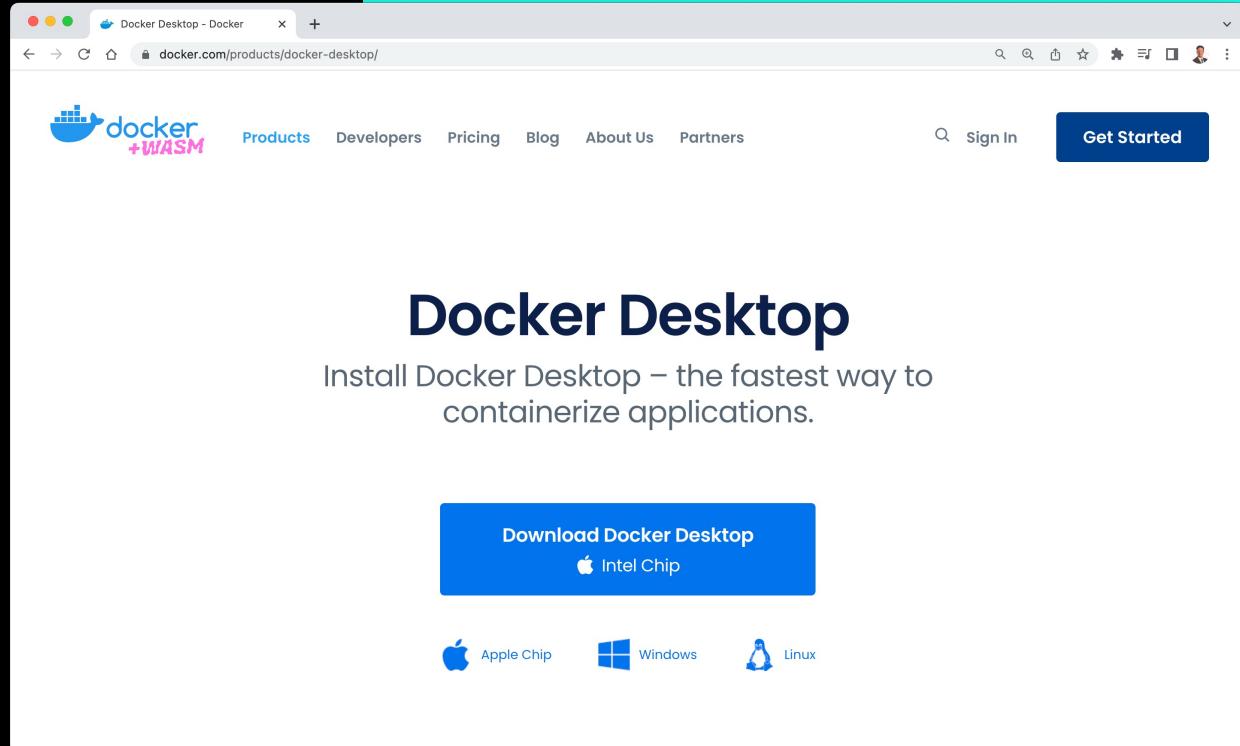


# Git Bash Installation

# (for Windows)



# Docker Desktop Installation



The screenshot shows the Docker Desktop product page on the official Docker website. The header includes the Docker logo and navigation links for Products, Developers, Pricing, Blog, About Us, and Partners. A search bar and sign-in options are also present. A prominent blue button at the top right says "Get Started". The main content features a large title "Docker Desktop" and a subtitle "Install Docker Desktop – the fastest way to containerize applications." Below this is a blue call-to-action button labeled "Download Docker Desktop" with the "Apple Intel Chip" option selected. Other download options for "Apple Chip", "Windows", and "Linux" are shown below.

Docker Desktop

Install Docker Desktop – the fastest way to containerize applications.

Download Docker Desktop

Apple Intel Chip

Apple Chip Windows Linux



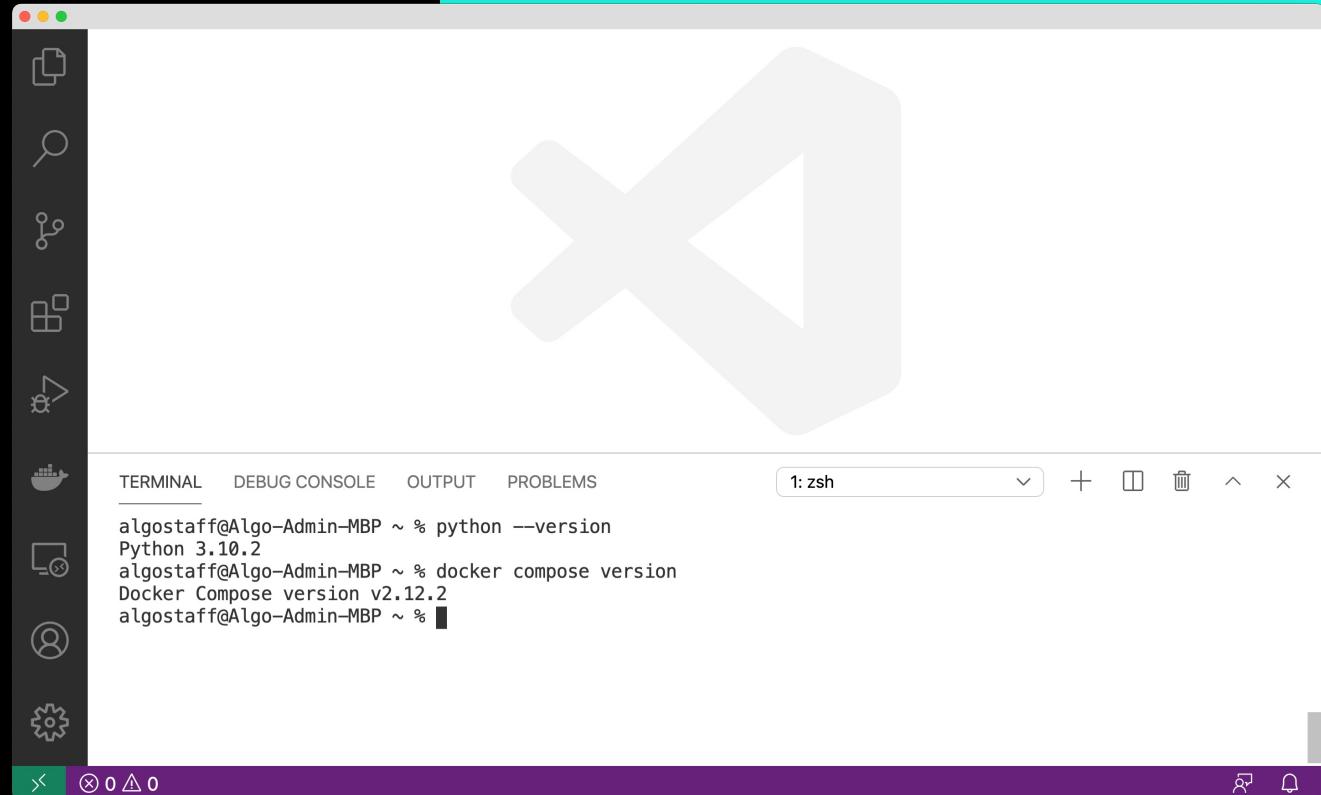
# Python Installation

$\geq 3.10$

The screenshot shows the Python.org website's download section. At the top, there's a navigation bar with tabs for Python, PSF, Docs, PyPI, Jobs, and Community. Below the navigation is the Python logo. To the right of the logo are buttons for Donate, Search, and Go, along with a Socialize link. A secondary navigation bar below the main one includes links for About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area features a large yellow button labeled "Download Python 3.11.0". Below this button, text encourages users to look for Python on other platforms like Windows, Linux/UNIX, macOS, and others. It also mentions ways to help test development versions. To the right of the text is a cartoon illustration of two boxes descending from the sky on yellow-and-white striped parachutes.



# Verify Prerequisites



A screenshot of the Visual Studio Code (VS Code) interface. The main area shows a large gray 'X' icon, indicating a problem or error. The left sidebar has a dark theme with white icons for file operations like copy, search, and refresh. The bottom bar features a purple footer with icons for close, minimize, maximize, and settings.

The terminal window displays the following command-line output:

```
algostaff@Algo-Admin-MBP ~ % python --version
Python 3.10.2
algostaff@Algo-Admin-MBP ~ % docker compose version
Docker Compose version v2.12.2
algostaff@Algo-Admin-MBP ~ %
```

The terminal tab is labeled "1: zsh".



# Developer Environment Components

Infrastructure

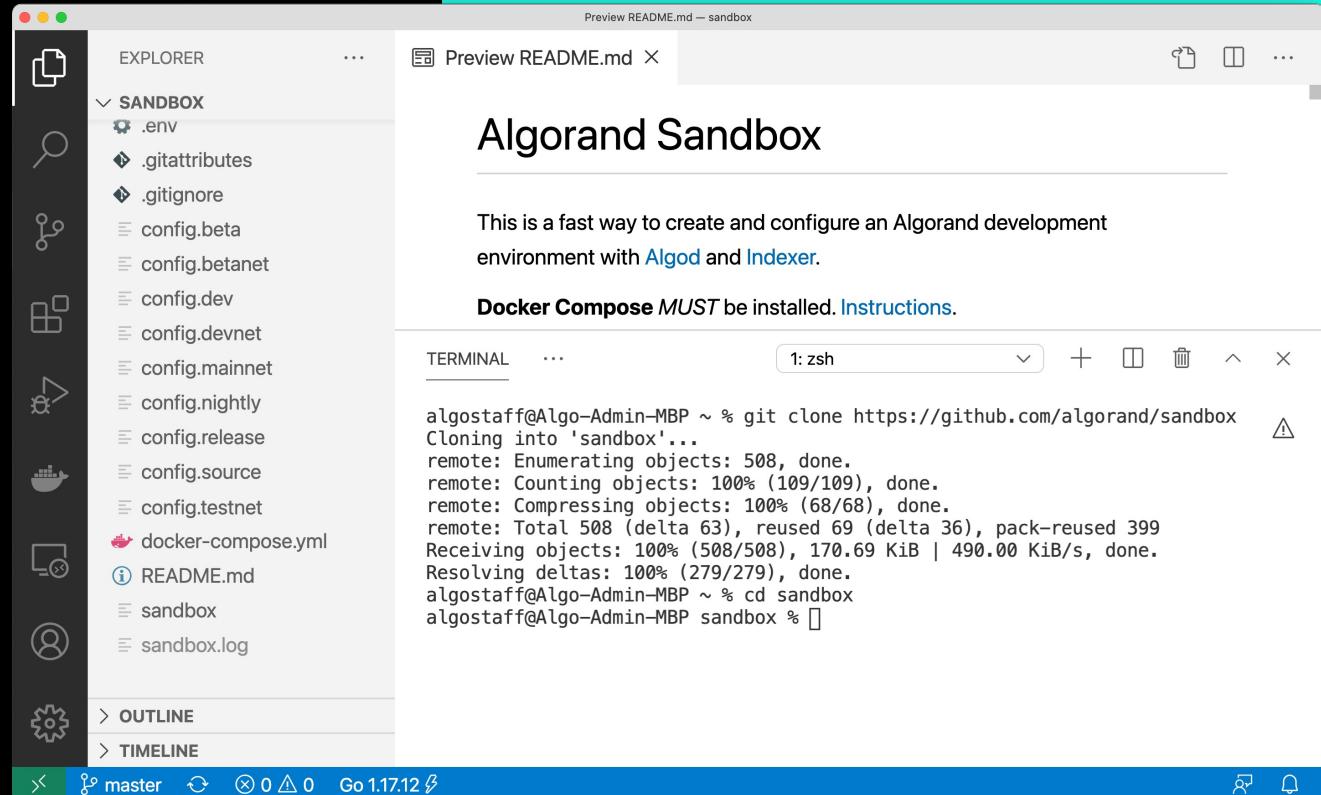


SDK

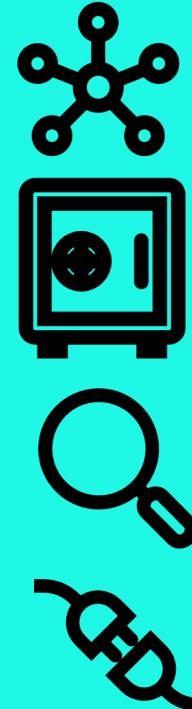
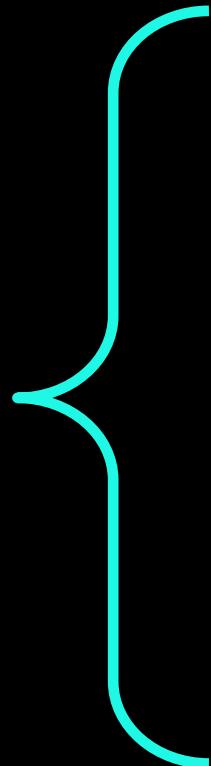
Language + Framework



# Sandbox Installation



# Sandbox Components



Network(s) 4001



Wallet 4002



Query Tool 8980



APIs ports



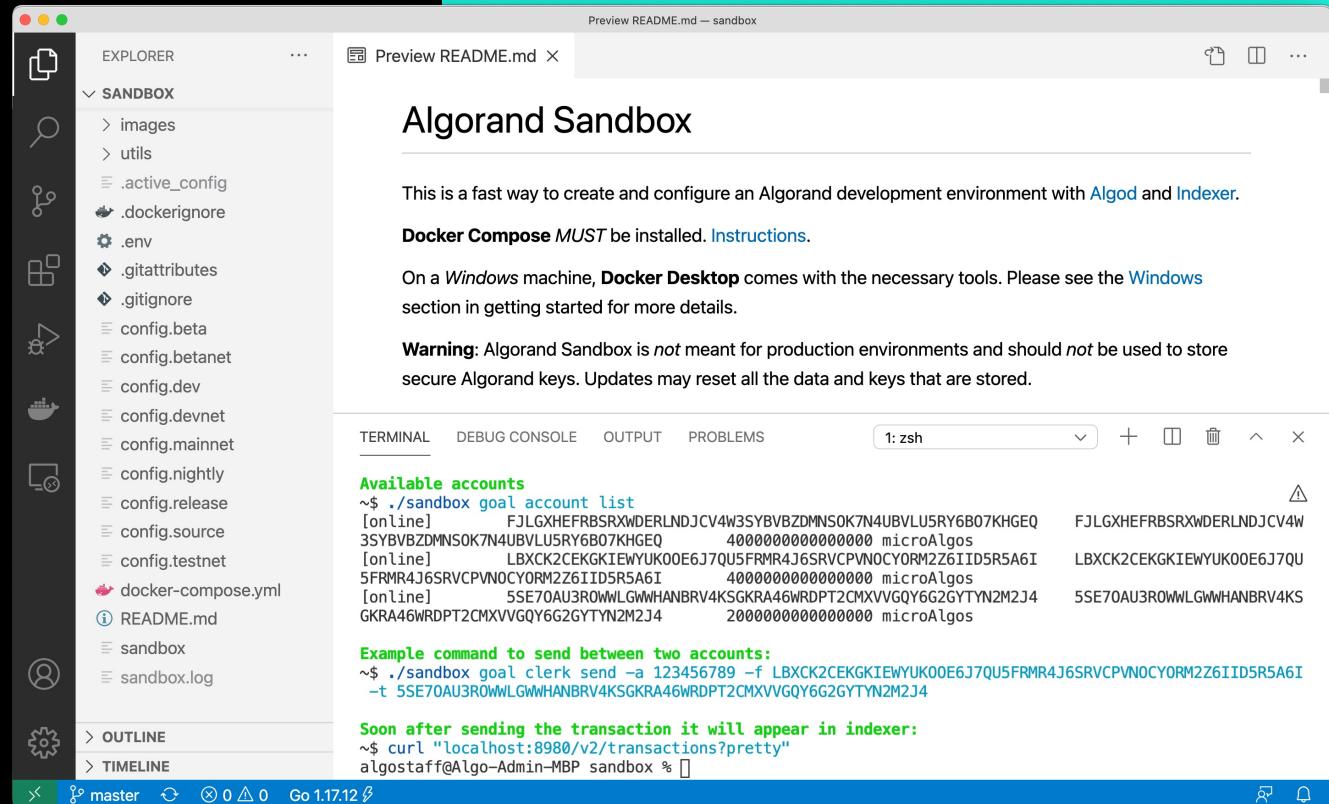
# Sandbox Execution



```
$ ./sandbox up dev -v. # start local private net
```



# Sandbox Execution



# Sandbox Commands



```
$ sandbox help  
$ sandbox status  
$ sandbox stop  
$ sandbox reset
```



# Developer Environment Components

Infrastructure



SDK



Language + Framework



# SDKs

The screenshot shows a web browser displaying the Algorand Developer Portal at [developer.algorand.org/docs/sdks/](https://developer.algorand.org/docs/sdks/). The page title is "Section Index - Algorand Dev". The main content area is titled "Algorand supported" and lists four programming languages with their respective logos and documentation links:

- Python**: Logo (yellow background with JS), [Python Section](#), [Python Repository](#), [Python Docs](#)
- JavaScript**: Logo (white background with blue and yellow JS), [JavaScript Section](#), [JavaScript Repository](#), [JavaScript Docs](#)
- Java**: Logo (white background with red Java), [Java Section](#), [Java Repository](#)
- Go**: Logo (blue background with white Go), [Go Section](#), [Go Repository](#), [Go Docs](#)

The left sidebar contains a navigation menu with the following items:

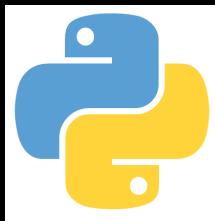
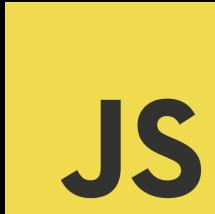
- Menu
- Algorand Developer Docs
  - > Get started
  - > Get details
  - SDKs
    - Section Index
  - > Python
  - > Javascript
  - > Go
  - > Java
  - > REST APIs
  - > CLI Tools
  - > Run a node
  - About
  - > Archive

The top right of the page features a "Build Learn Discover Connect Challenges" navigation bar, a "Sign In" button, and a "Sign Up" button. A "Table of contents" link is located on the right side of the main content area.



#Decipher2022

# SDK Installations



```
$ npm install algosdk
```

```
$ pip install py-algorand-sdk
```

# Developer Environment Components

Infrastructure



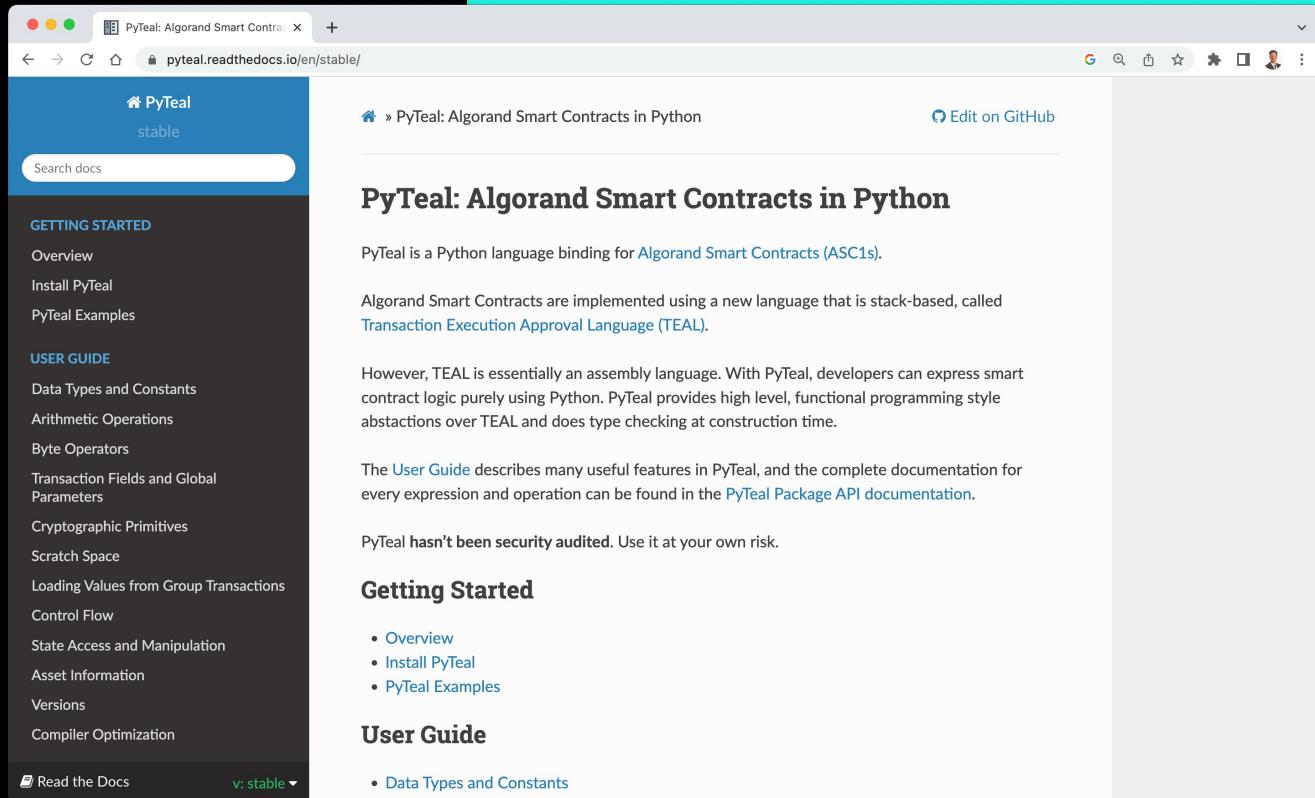
SDK



Language + Framework



# Language



The screenshot shows a web browser displaying the PyTeal documentation at [pyteal.readthedocs.io/en/stable/](https://pyteal.readthedocs.io/en/stable/). The page has a dark blue header with the PyTeal logo and the word "stable". A sidebar on the left contains navigation links for "GETTING STARTED" (Overview, Install PyTeal, PyTeal Examples) and "USER GUIDE" (Data Types and Constants, Arithmetic Operations, Byte Operators, Transaction Fields and Global Parameters, Cryptographic Primitives, Scratch Space, Loading Values from Group Transactions, Control Flow, State Access and Manipulation, Asset Information, Versions, Compiler Optimization). At the bottom of the sidebar are "Read the Docs" and a dropdown menu showing "v: stable". The main content area features a title "PyTeal: Algorand Smart Contracts in Python" and a paragraph explaining that PyTeal is a Python language binding for Algorand Smart Contracts (ASC1s). It notes that Algorand Smart Contracts are implemented using TEAL, a stack-based assembly language, and that PyTeal provides a high-level, functional programming style abstraction over TEAL. The User Guide section describes various features and operations, and a note states that PyTeal hasn't been security audited.

PyTeal: Algorand Smart Contracts in Python

PyTeal is a Python language binding for [Algorand Smart Contracts \(ASC1s\)](#).

Algorand Smart Contracts are implemented using a new language that is stack-based, called [Transaction Execution Approval Language \(TEAL\)](#).

However, TEAL is essentially an assembly language. With PyTeal, developers can express smart contract logic purely using Python. PyTeal provides high level, functional programming style abstractions over TEAL and does type checking at construction time.

The [User Guide](#) describes many useful features in PyTeal, and the complete documentation for every expression and operation can be found in the [PyTeal Package API documentation](#).

PyTeal hasn't been security audited. Use it at your own risk.

## Getting Started

- [Overview](#)
- [Install PyTeal](#)
- [PyTeal Examples](#)

## User Guide

- [Data Types and Constants](#)



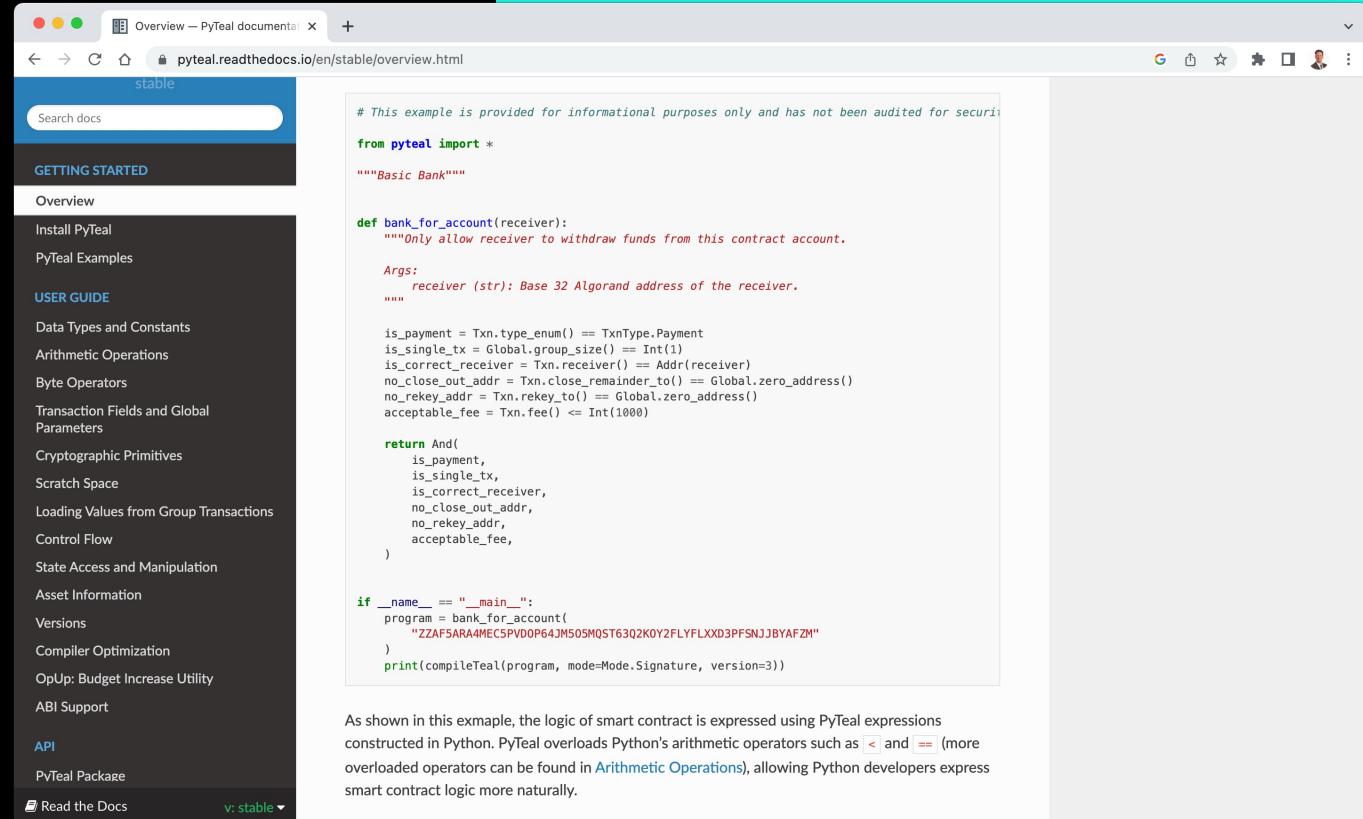
# PyTeal Installation



```
$ pip install pyteal
```



# PyTeal Syntax



The screenshot shows a web browser displaying the PyTeal documentation at [pyteal.readthedocs.io/en/stable/overview.html](https://pyteal.readthedocs.io/en/stable/overview.html). The page title is "Overview — PyTeal documentation". The left sidebar contains a search bar and a navigation menu with sections like "GETTING STARTED", "USER GUIDE", and "API". The main content area shows a Python code example for a "Basic Bank" smart contract. The code defines a function `bank_for_account` that checks if a transaction is a payment, has one receiver, and is sent to the correct address. It also ensures no close-out address is used and the fee is acceptable. The code then prints the compiled Teal program.

```
# This example is provided for informational purposes only and has not been audited for security.
from pyteal import *

"""Basic Bank"""

def bank_for_account(receiver):
    """Only allow receiver to withdraw funds from this contract account.

    Args:
        receiver (str): Base 32 Algorand address of the receiver.
    """

    is_payment = Txn.type_enum() == TxnType.Payment
    is_single_tx = Global.group_size() == Int(1)
    is_correct_receiver = Txn.receiver() == Addr(receiver)
    no_close_out_addr = Txn.close_remainder_to() == Global.zero_address()
    no_rekey_addr = Txn.rekey_to() == Global.zero_address()
    acceptable_fee = Txn.fee() <= Int(1000)

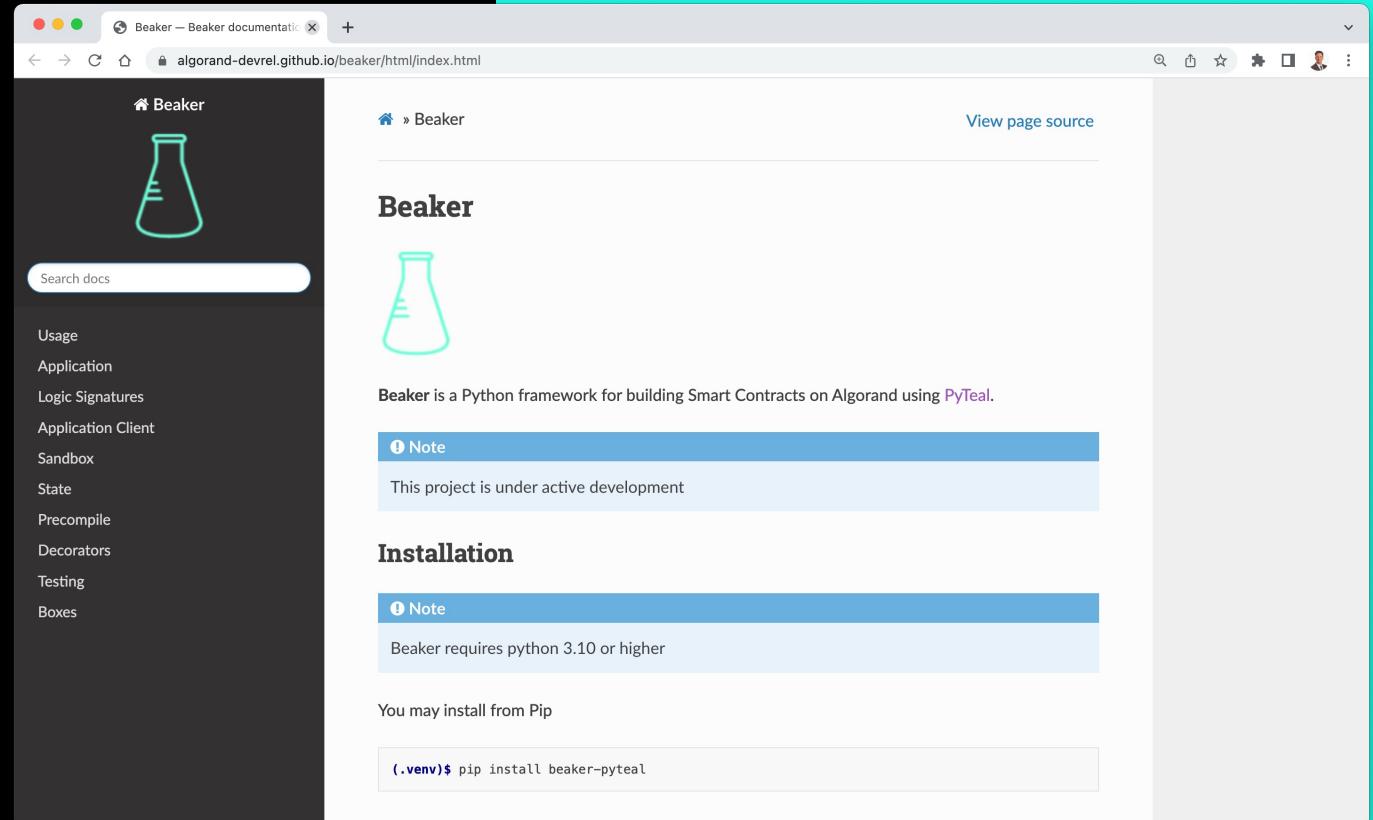
    return And(
        is_payment,
        is_single_tx,
        is_correct_receiver,
        no_close_out_addr,
        no_rekey_addr,
        acceptable_fee,
    )

if __name__ == "__main__":
    program = bank_for_account(
        "ZAF5ARA4MEC5PV0D64JM505MQST63Q2K0Y2FLYFLXX03PFSNJJBYAFZM"
    )
    print(compileTeal(program, mode=Mode.Signature, version=3))
```

As shown in this example, the logic of smart contract is expressed using PyTeal expressions constructed in Python. PyTeal overloads Python's arithmetic operators such as `<` and `==` (more overloaded operators can be found in [Arithmetic Operations](#)), allowing Python developers express smart contract logic more naturally.



# Framework



The screenshot shows a web browser displaying the Beaker documentation at [algorand-devrel.github.io/beaker/html/index.html](https://algorand-devrel.github.io/beaker/html/index.html). The page has a dark header with a light gray sidebar on the left containing a search bar and a list of links: Usage, Application, Logic Signatures, Application Client, Sandbox, State, Precompile, Decorators, Testing, and Boxes. The main content area features a large green beaker icon and a brief introduction: "Beaker is a Python framework for building Smart Contracts on Algorand using PyTeal." It includes two "Note" callouts: one stating "This project is under active development" and another stating "Beaker requires python 3.10 or higher". Below these notes, there's a section on "Installation" with a note about Python requirements.

Beaker

Search docs

Usage

Application

Logic Signatures

Application Client

Sandbox

State

Precompile

Decorators

Testing

Boxes

» Beaker

View page source

## Beaker

Beaker is a Python framework for building Smart Contracts on Algorand using [PyTeal](#).

**Note**

This project is under active development

**Note**

Beaker requires python 3.10 or higher

You may install from Pip

```
(.venv)$ pip install beaker-pyteal
```



# Beaker Installation

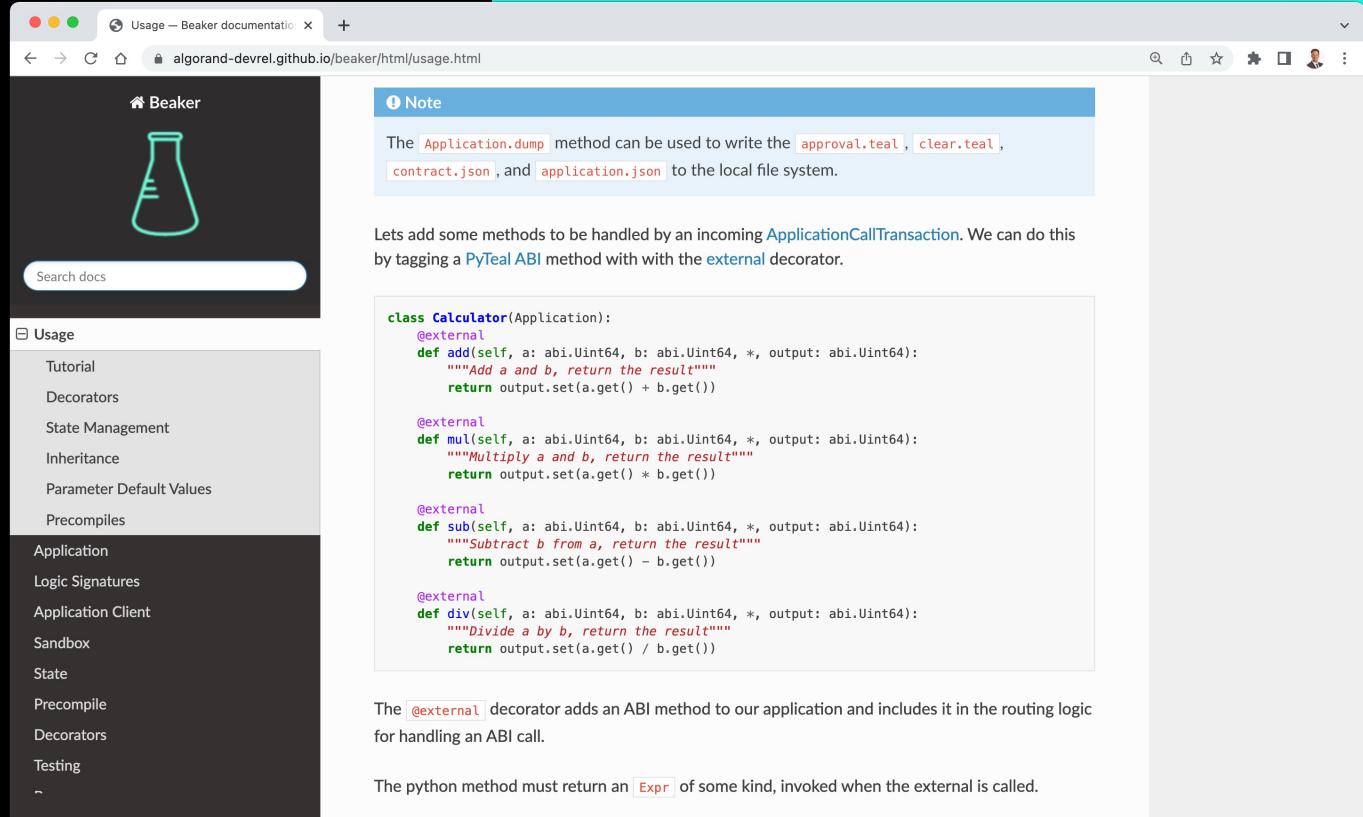


Beaker

( auto installs pyteal )

```
$ pip install beaker-pyteal
```

# Beaker Syntax



The screenshot shows a web browser displaying the Beaker documentation at [algorand-devrel.github.io/beaker/html/usage.html](https://algorand-devrel.github.io/beaker/html/usage.html). The page has a dark theme with a light sidebar. The sidebar contains a logo of a flask labeled "Beaker" and a search bar. The main content area has a blue header bar with the text "Note". Below it, a note explains that the `Application.dump` method can be used to write files like `approval.teal`, `clear.teal`, `contract.json`, and `application.json` to the local file system. The main text discusses adding methods to handle `ApplicationCallTransaction` using the `@external` decorator. It provides a code example for a `Calculator` application:

```
class Calculator(Application):
    @external
    def add(self, a: abi.Uint64, b: abi.Uint64, *, output: abi.Uint64):
        """Add a and b, return the result"""
        return output.set(a.get() + b.get())

    @external
    def mul(self, a: abi.Uint64, b: abi.Uint64, *, output: abi.Uint64):
        """Multiply a and b, return the result"""
        return output.set(a.get() * b.get())

    @external
    def sub(self, a: abi.Uint64, b: abi.Uint64, *, output: abi.Uint64):
        """Subtract b from a, return the result"""
        return output.set(a.get() - b.get())

    @external
    def div(self, a: abi.Uint64, b: abi.Uint64, *, output: abi.Uint64):
        """Divide a by b, return the result"""
        return output.set(a.get() / b.get())
```

The note below the code explains that the `@external` decorator adds an ABI method to the application and includes it in the routing logic for handling an ABI call. A final note states that the Python method must return an `Expr` of some kind, invoked when the external is called.



# Getting Started

The screenshot shows a browser window displaying the `README.md` file from the `algorand-devrel/workshop-gettingstarted` repository on GitHub. The page contains three main sections: **build unsigned transaction**, **sign transaction**, and **submit transaction**. Each section includes a code snippet demonstrating the corresponding step using the Algorand Python client library.

**build unsigned transaction**

```
params = algod_client.suggested_params()  
receiver = addr2  
note = "Hello World".encode()  
amount = 1000000  
unsigned_txn = transaction.PaymentTxn(addr1, params, receiver, amount)
```

**sign transaction**

```
signed_txn = unsigned_txn.sign(wallet.export_key(addr1))
```

**submit transaction**

```
txid = algod_client.send_transaction(signed_txn)  
print("Successfully sent transaction with txID: {}".format(txid))
```



# Developer Environment Components

Infrastructure



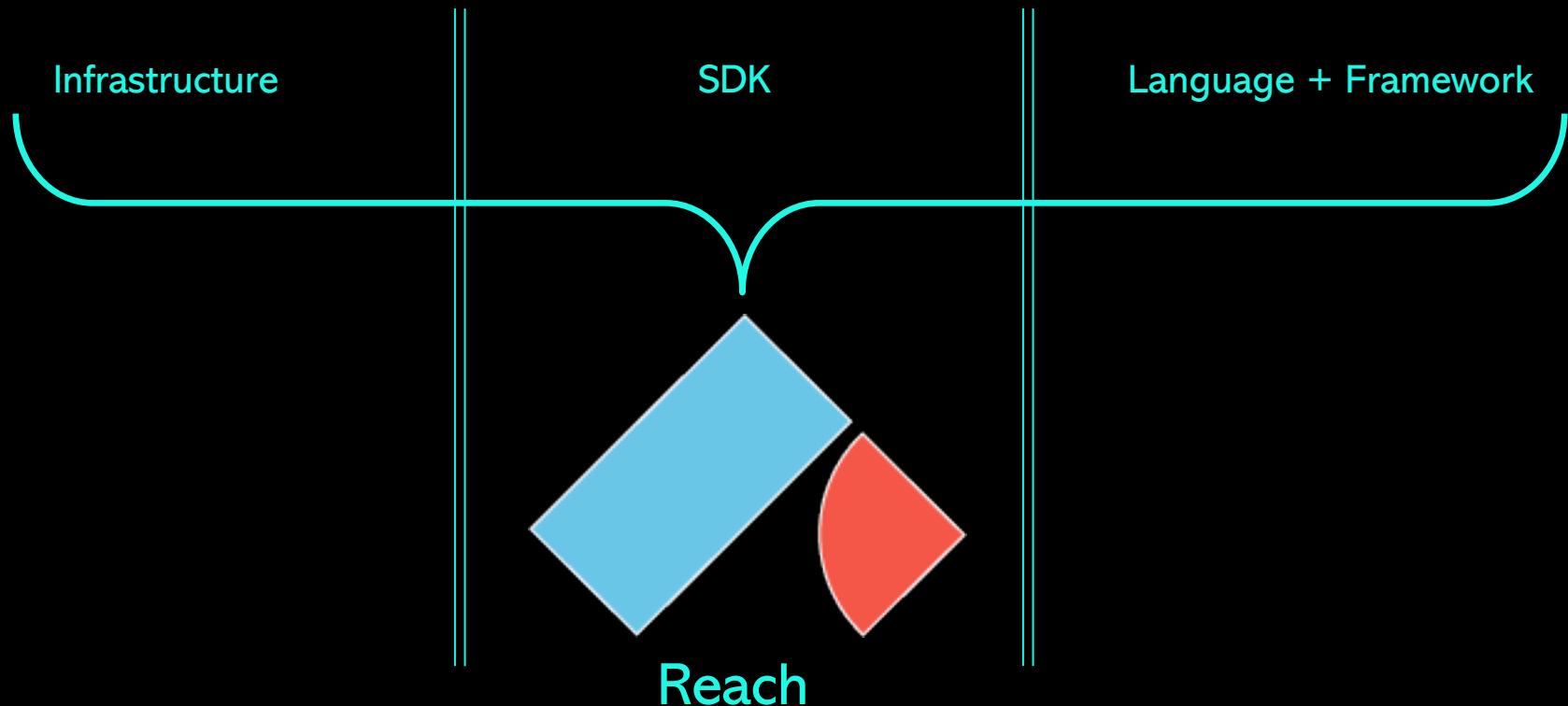
SDK



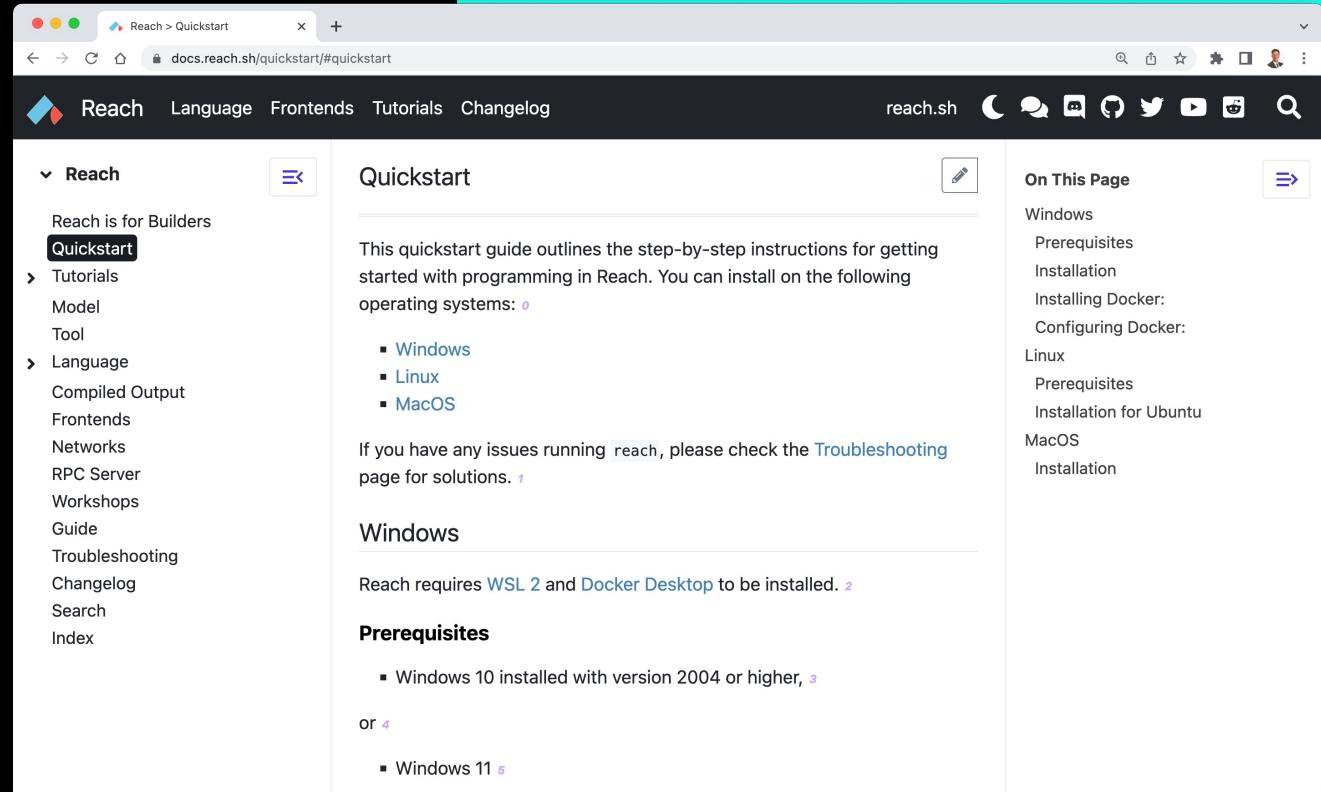
Language + Framework



# Developer Environment Components



# Reach Installation



The screenshot shows a web browser displaying the Reach Quickstart guide at [docs.reach.sh/quickstart/#quickstart](https://docs.reach.sh/quickstart/#quickstart). The page has a dark theme with a sidebar on the left and a main content area on the right.

**Left Sidebar:**

- Reach is for Builders
- Quickstart** (highlighted)
- Tutorials
  - Model
  - Tool
- Language
  - Compiled Output
  - Frontends
  - Networks
  - RPC Server
  - Workshops
  - Guide
  - Troubleshooting
  - Changelog
  - Search
  - Index

**Main Content Area:**

## Quickstart

This quickstart guide outlines the step-by-step instructions for getting started with programming in Reach. You can install on the following operating systems: [Windows](#), [Linux](#), and [MacOS](#).

If you have any issues running `reach`, please check the [Troubleshooting](#) page for solutions. [1](#)

### Windows

Reach requires [WSL 2](#) and [Docker Desktop](#) to be installed. [2](#)

### Prerequisites

- Windows 10 installed with version 2004 or higher, [3](#)

or [4](#)

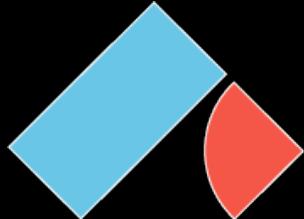
- Windows 11 [5](#)

**On This Page**

- Windows
- Prerequisites
- Installation
  - Installing Docker:
    - Configuring Docker:
  - Linux
    - Prerequisites
    - Installation for Ubuntu
  - MacOS
    - Installation



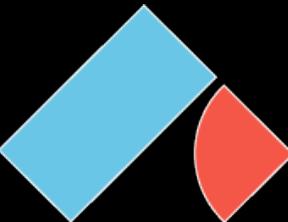
# Reach Installation



```
$ sudo apt install make curl  
$ mkdir -p ~/reach && cd ~/reach  
$ curl https://docs.reach.sh/reach -o reach ; chmod +x reach  
$ ./reach version
```



# Reach Syntax



The main part of the program looks like this: [12](#)

```
/examples/overview/index.rsh
```

```
1 'reach 0.1';
2 'use strict';
3
4 export const main = Reach.App(() => {
5   const A = Participant('Alice', {
6     request: UInt,
7     info: Bytes(128),
8   });
9   const B = Participant('Bob', {
10    want: Fun([UInt], Null),
11    got: Fun([Bytes(128)], Null),
12  });
13   init();
14
15   A.only(() => {
```

- Line 1 specifies that this is a Reach program. [13](#)
- Line 2 specifies that this program will be compiled with strict mode, which enables unused variable checks. [14](#)
- Line 4 defines the main export from this program. `main` is the default used by Reach. [15](#)
- Line 4 also specifies that it is an application. [16](#)
- Line 5 specifies that the program identifier `A` will represent the Alice

## Reach

Reach is for Builders

Quickstart

### Tutorials

#### Overview

Wisdom For Sale

➤ Rock, Paper, Scissors!

Répondez S'il Vous Plaît

Model

Tool

### Language

Compiled Output

Frontends

Networks

RPC Server

Workshops

Guide

Troubleshooting

Changelog

Search

Index

reach.sh

## On This Page

Decentralized applications

### A minimal Reach program

Compile

Verify

Interface

Execute

Web app

Next steps

#Decipher2022

52

# Developer Portal

The screenshot shows the Algorand Developer Portal homepage. At the top, there's a navigation bar with links for Build, Learn, Discover, Connect, and Challenges. A search bar is also present. On the left side, a large call-to-action section features the text "Build the future on Algorand" and a bulleted list: "Defi at global scale", "Rapidly growing ecosystem", and "Sub 5-second finality, low fees". On the right side, a modal window titled "Welcome to Algorand Developer Portal!" displays the message "Let's get started..." and a category selection interface. The categories listed are: 1 Blockchain basics, 2 Smart contracts & dApps, 3 Tokenization, 4 Integration, and 5 See all. There are also up and down arrow buttons for navigating through the categories.



<https://developer.algorand.org/>



#Decipher2022

# Developer Portal



The screenshot shows a web browser window for the Algorand Developer Portal at [developer.algorand.org/challenges/transaction-rookie/?language=python](https://developer.algorand.org/challenges/transaction-rookie/?language=python). The main page displays a challenge titled "Payment Transaction Rookie" under the "Algo" category. The challenge description is "Send a basic payment transaction". It shows that 22 people have acquired it, and the asset ID is 801348719. A modal window titled "Payment Transaction Rookie" is open, indicating success with a trophy icon and the message "Congratulations! You have successfully completed the Payment Transaction Rookie challenge. You can now claim your on-chain badge reward and receive it in the wallet of your choice!". A "Claim Badge" button is present in the modal. Below the modal, a terminal-like interface shows the command "confirmation(client," followed by a progress bar and the text "Transactions validated! Collect your badge :)".

#Decipher2022

# Show me the code



#Decipher2022

# #THANKS



@ryanRfox



Ryan R. Fox

Developer Advocate

@Algorand

## Let's Connect

#Decipher2022

