LAYER TWO AMSTERDAM

DEVCONNECT AMS

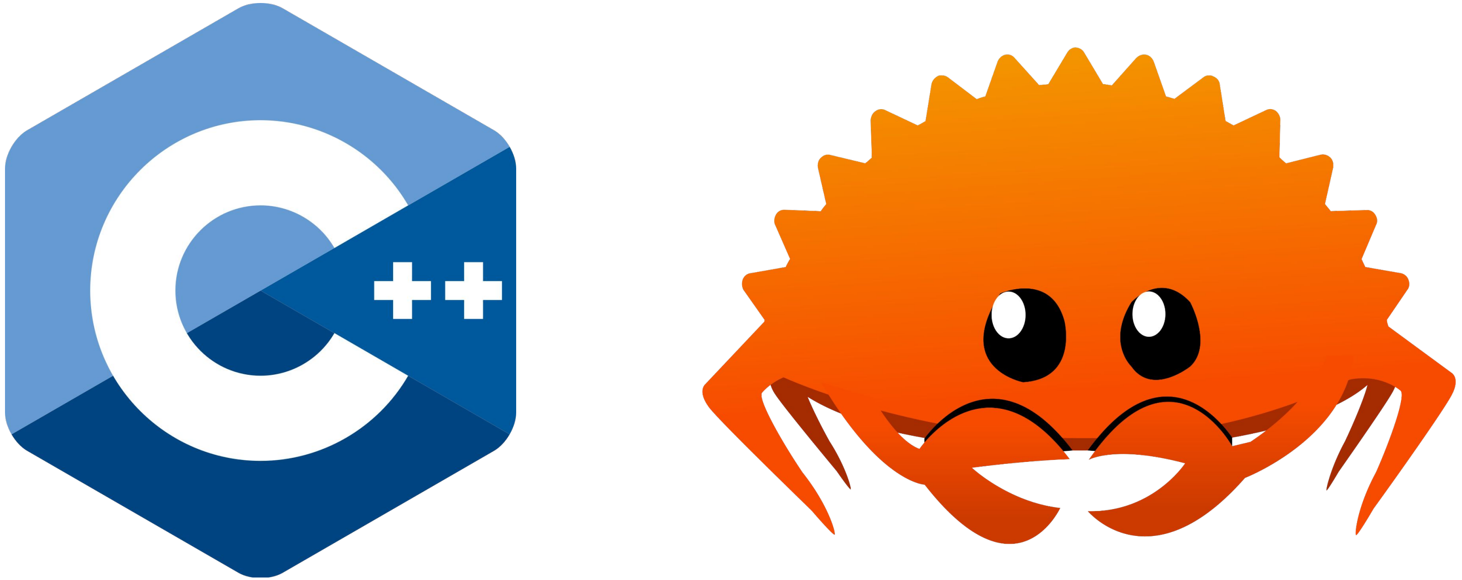
L2BEAT

Emily Herbert

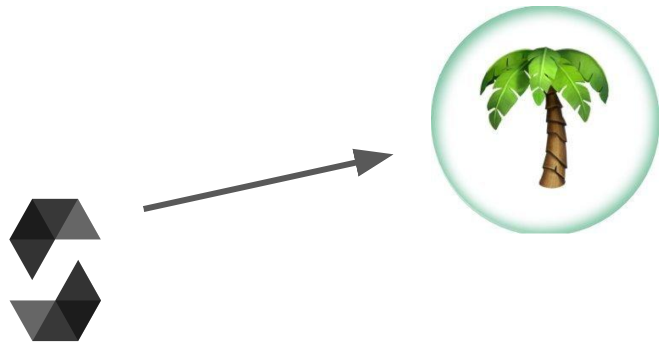
Developing Smart Contracts in Sway

Fuel Labs

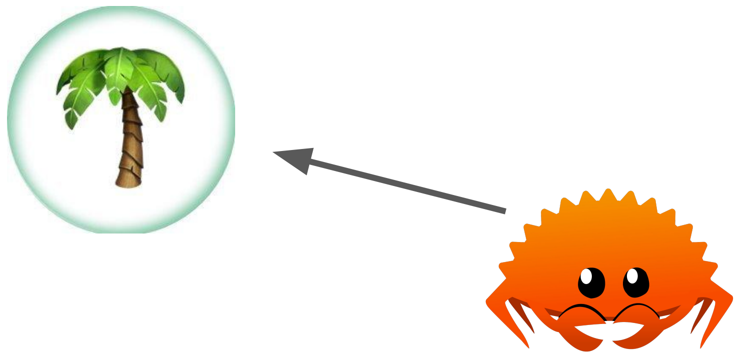
STARKWARE / ARBITRUM / OPTIMISM / BOBA NETWORK / METIS / FUEL LABS / NERVOS / ZKSYNC



What is Sway?



+ “domain-specific”
+ first-class smart contract and blockchain support
+ contract storage as language construct
+ sharable contract interfaces



+ “feels like Rust”
+ static types with trait-based inheritance
+ verbose and friendly compiler
+ packaged with tooling
+ team organization

⚡ in active development ⚡

Sway 🌴

smart-contract language

↓

FuelVM

blockchain-optimized VM

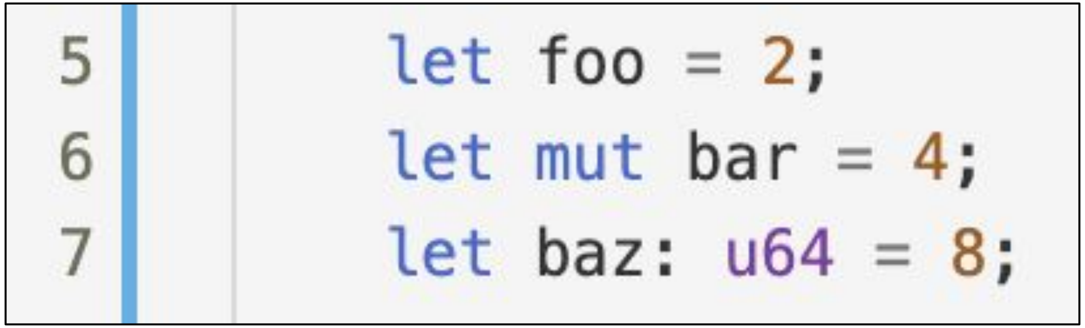
↓

Fuel ⚡

the fastest modular execution layer

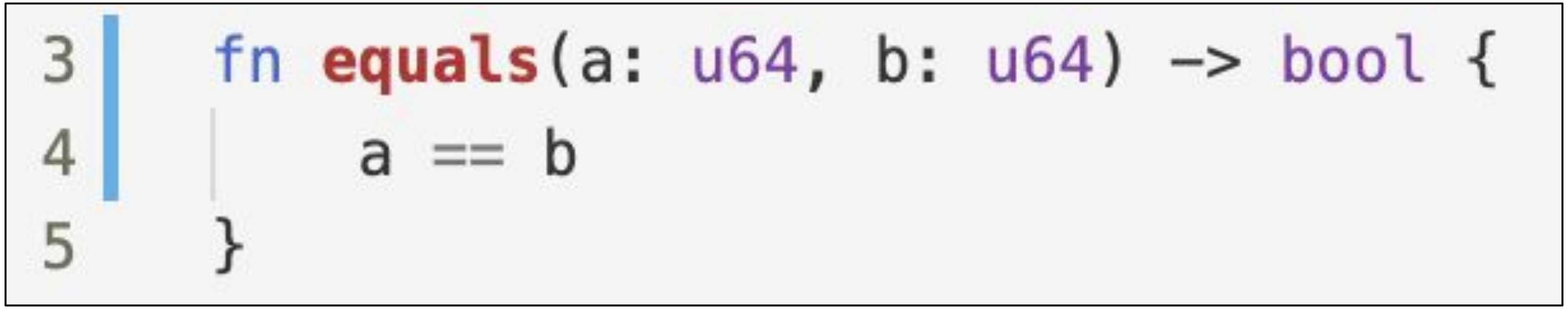
Sway feels like Rust?

variables:



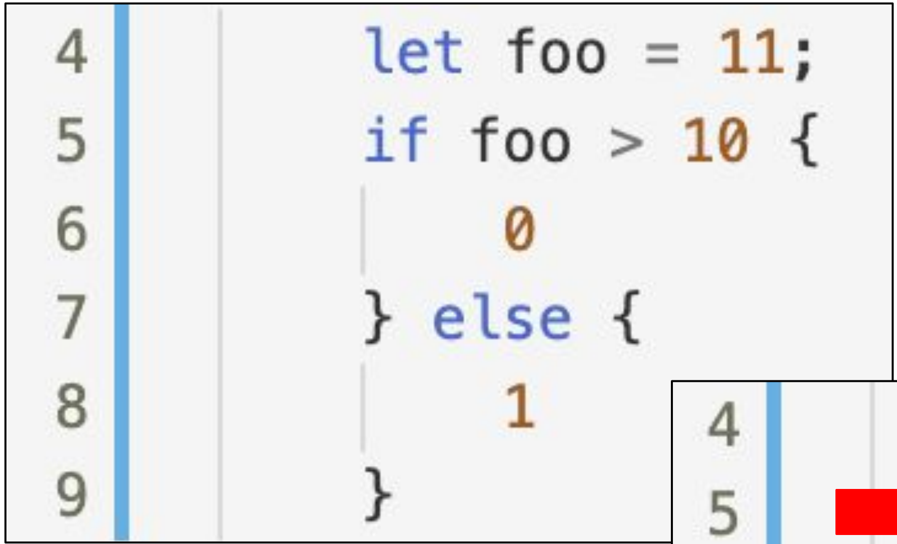
```
        let foo = 2;
        let mut bar = 4;
        let baz: u64 = 8;
```

functions:

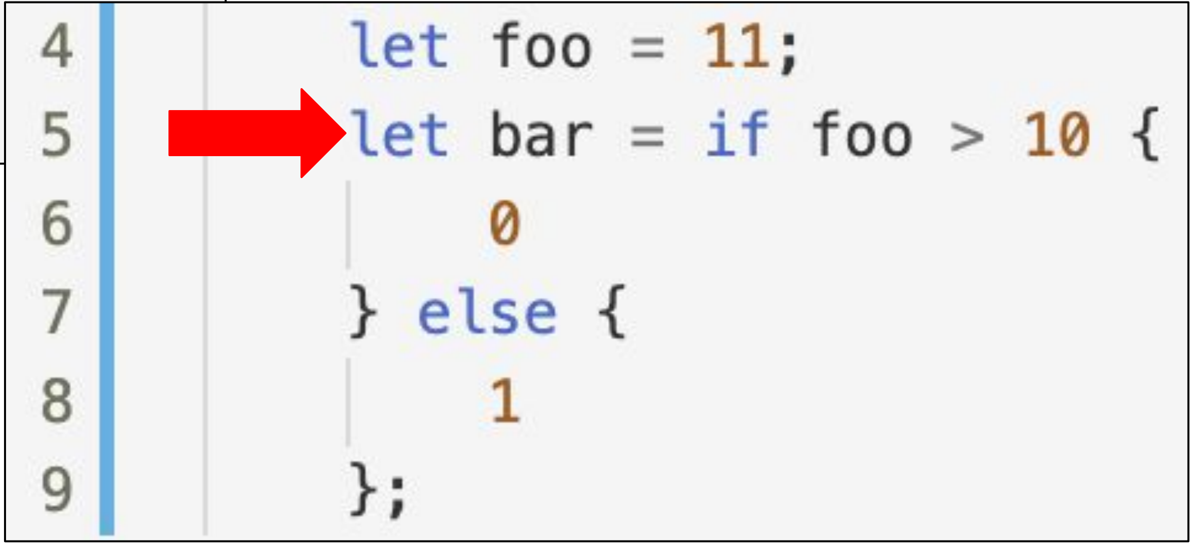


```
fn equals(a: u64, b: u64) -> bool {
    a == b
}
```

if expressions:

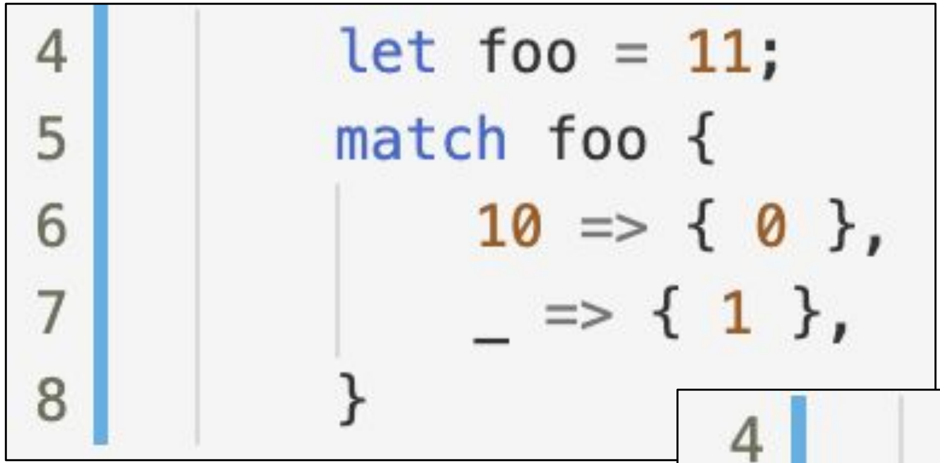


```
let foo = 11;
if foo > 10 {
    0
} else {
    1
}
```

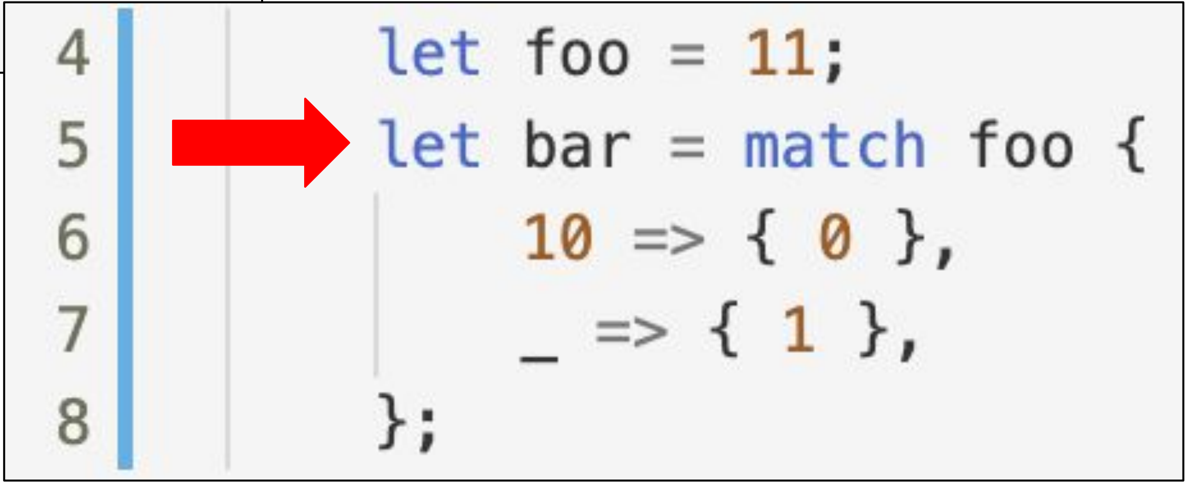


```
let foo = 11;
let bar = if foo > 10 {
    0
} else {
    1
};
```

match expressions:

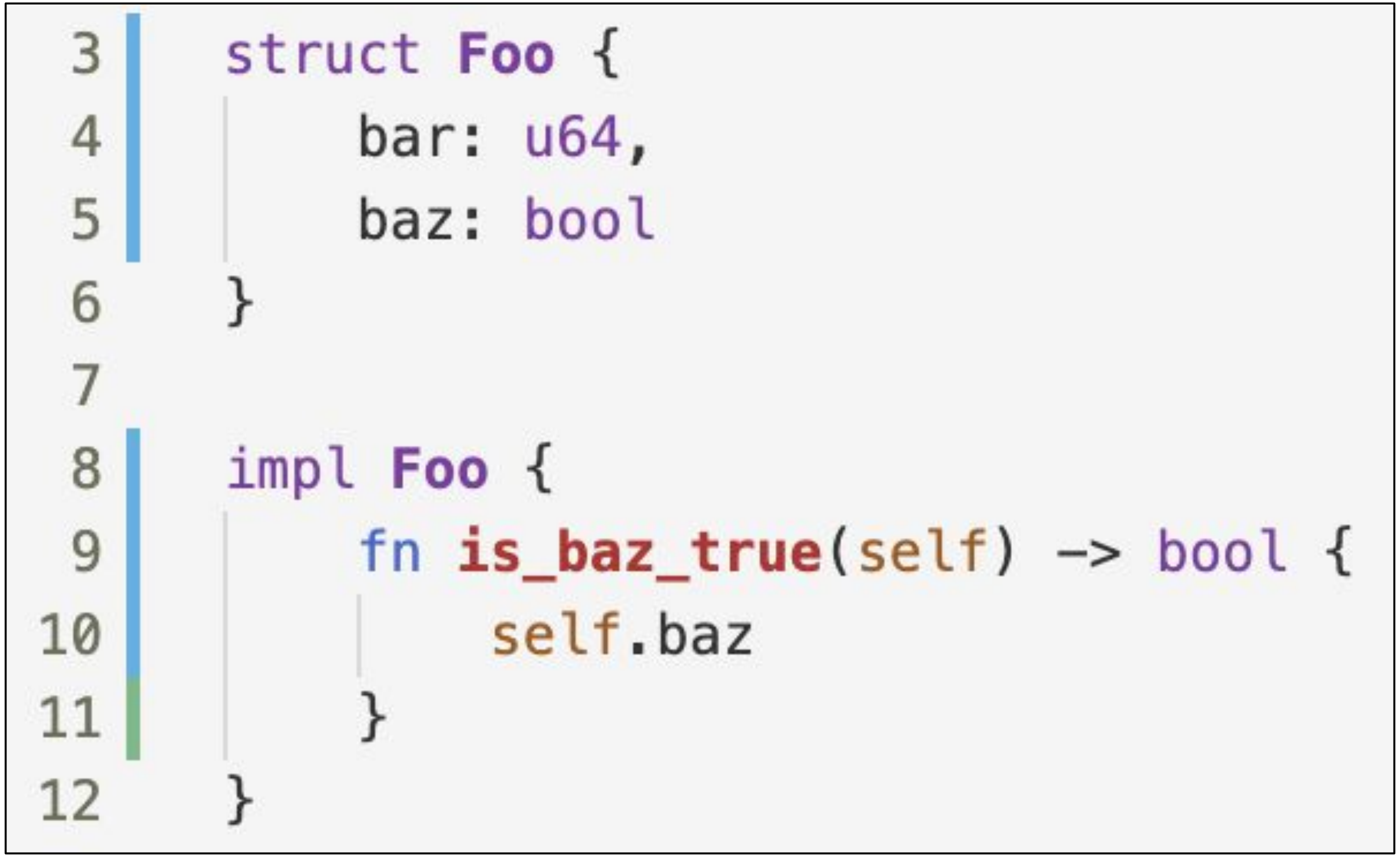


```rust
let foo = 11;
match foo {
    10 => { 0 },
    _ => { 1 },
}
```



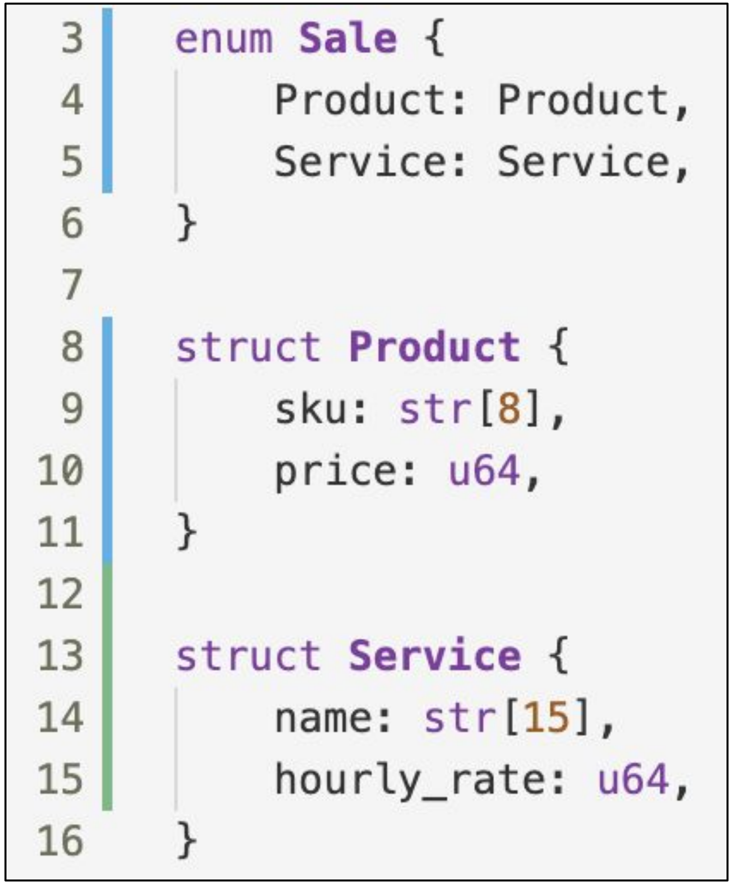
```rust
let foo = 11;
let bar = match foo {
    10 => { 0 },
    _ => { 1 },
};
```

structs (with methods):



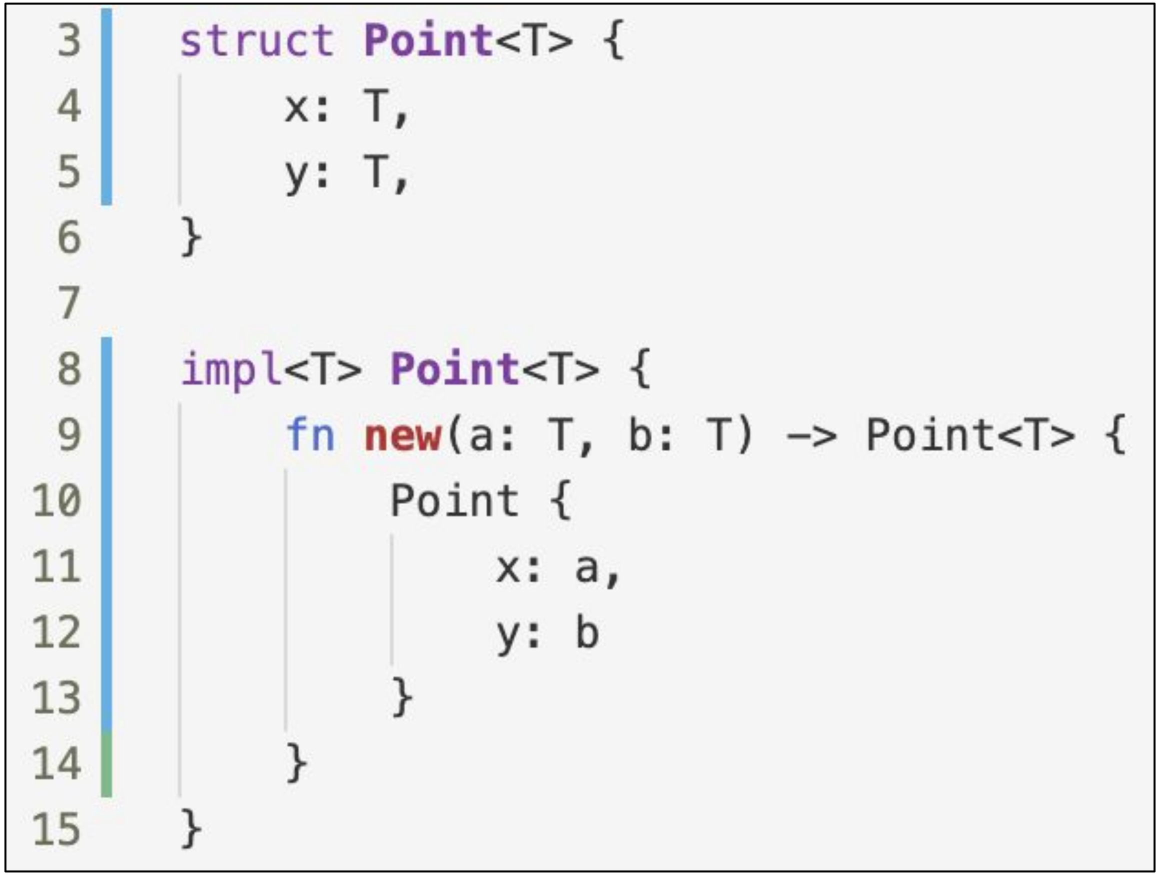
```rust
struct Foo {
    bar: u64,
    baz: bool
}

impl Foo {
    fn is_baz_true(self) -> bool {
        self.baz
    }
}
```

enums (sum types):



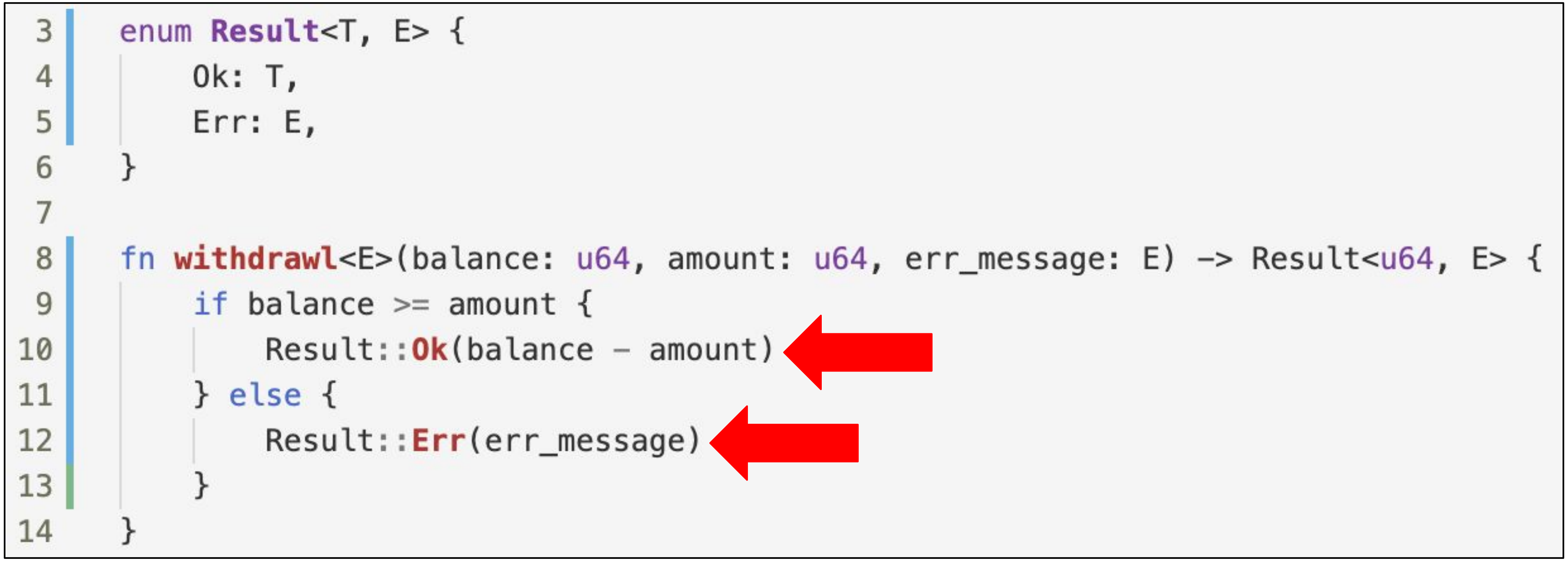
```
enum Sale {
    Product: Product,
    Service: Service,
}

struct Product {
    sku: str[8],
    price: u64,
}

struct Service {
    name: str[15],
    hourly_rate: u64,
}
```

generic types and type inference:



```rust
struct Point<T> {
    x: T,
    y: T,
}

impl<T> Point<T> {
    fn new(a: T, b: T) -> Point<T> {
        Point {
            x: a,
            y: b
        }
    }
}
```

error handling:



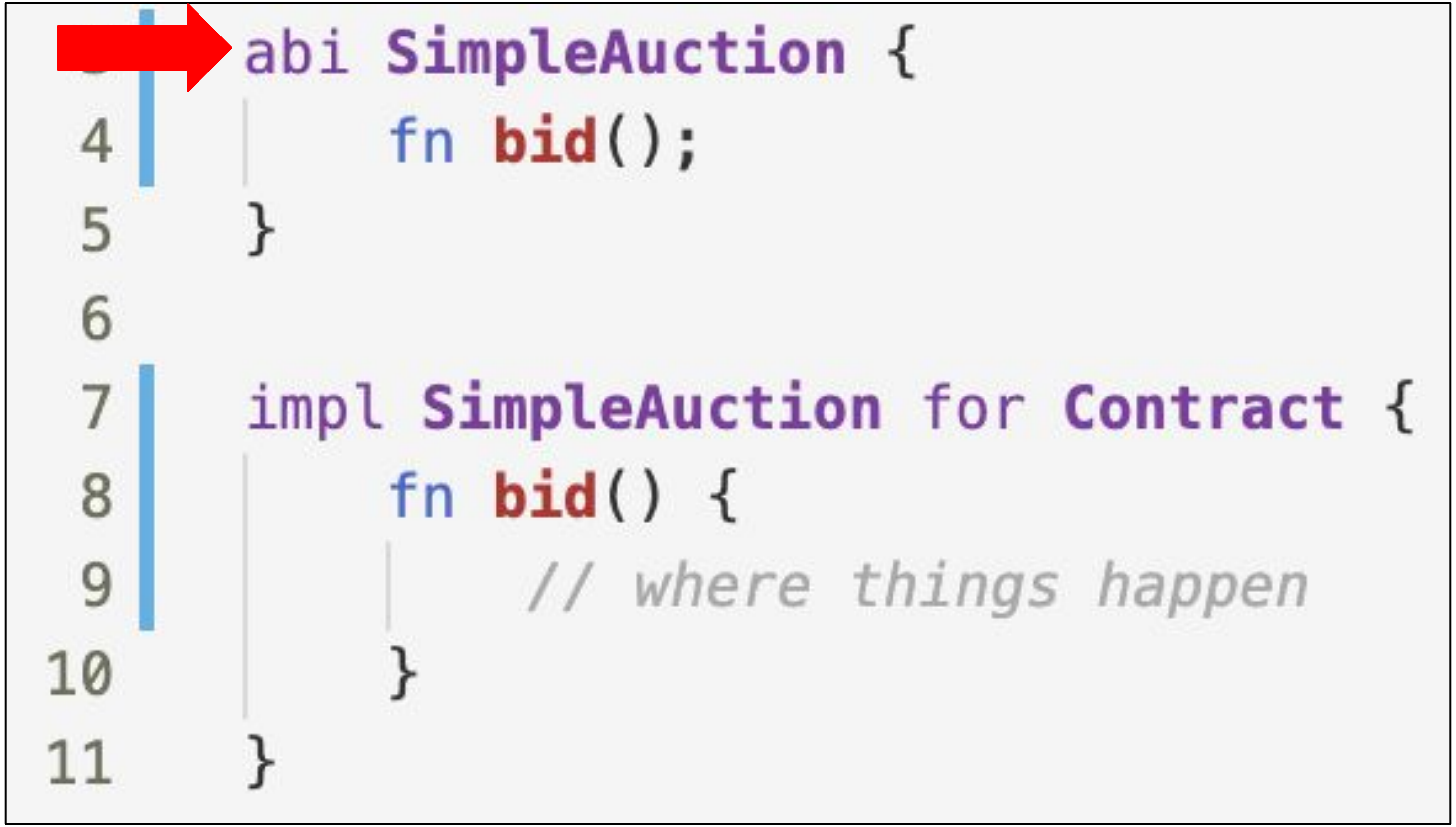
```
enum Result<T, E> {
    Ok: T,
    Err: E,
}

fn withdrawl<E>(balance: u64, amount: u64, err_message: E) -> Result<u64, E> {
    if balance >= amount {
        Result::Ok(balance - amount)
    } else {
        Result::Err(err_message)
    }
}
```

Sway is domain-specific?

Sway primitive types:

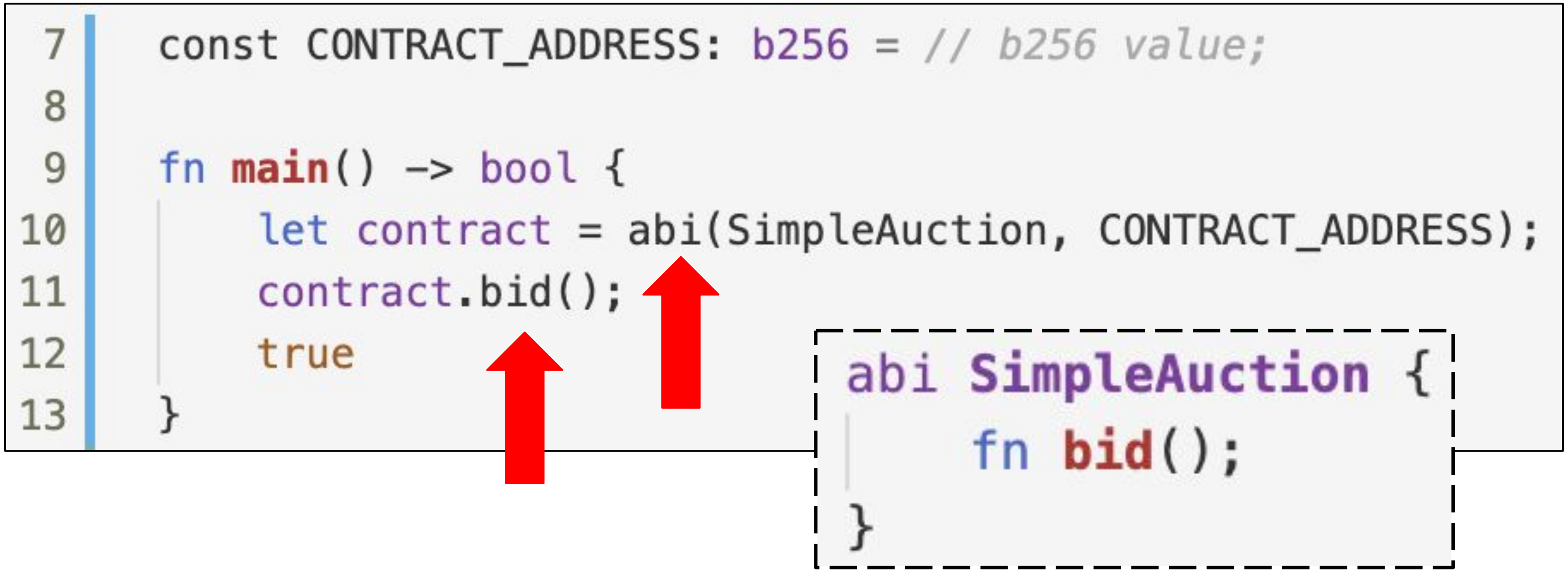
1. unsigned integers
2. booleans
3. static-length strings
4. 32-byte values
5. single-byte values
6. the unit type

first-class contracts:



```
abi SimpleAuction {
    fn bid();
}

impl SimpleAuction for Contract {
    fn bid() {
        // where things happen
    }
}
```

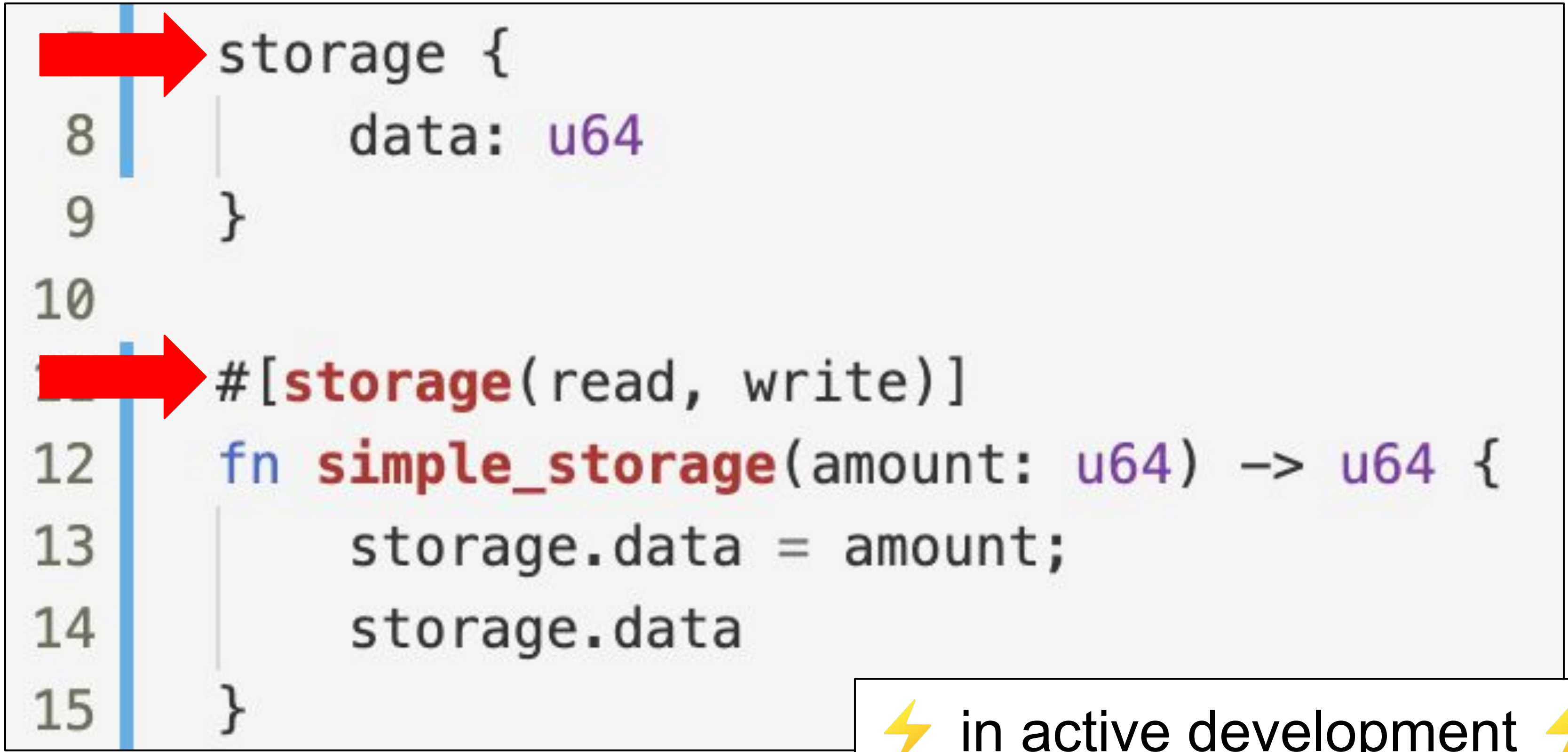
calling contracts:



```
const CONTRACT_ADDRESS: b256 = // b256 value;

fn main() -> bool {
    let contract = abi(SimpleAuction, CONTRACT_ADDRESS);
    contract.bid();
    true
}
```

```
abi SimpleAuction {
    fn bid();
}
```

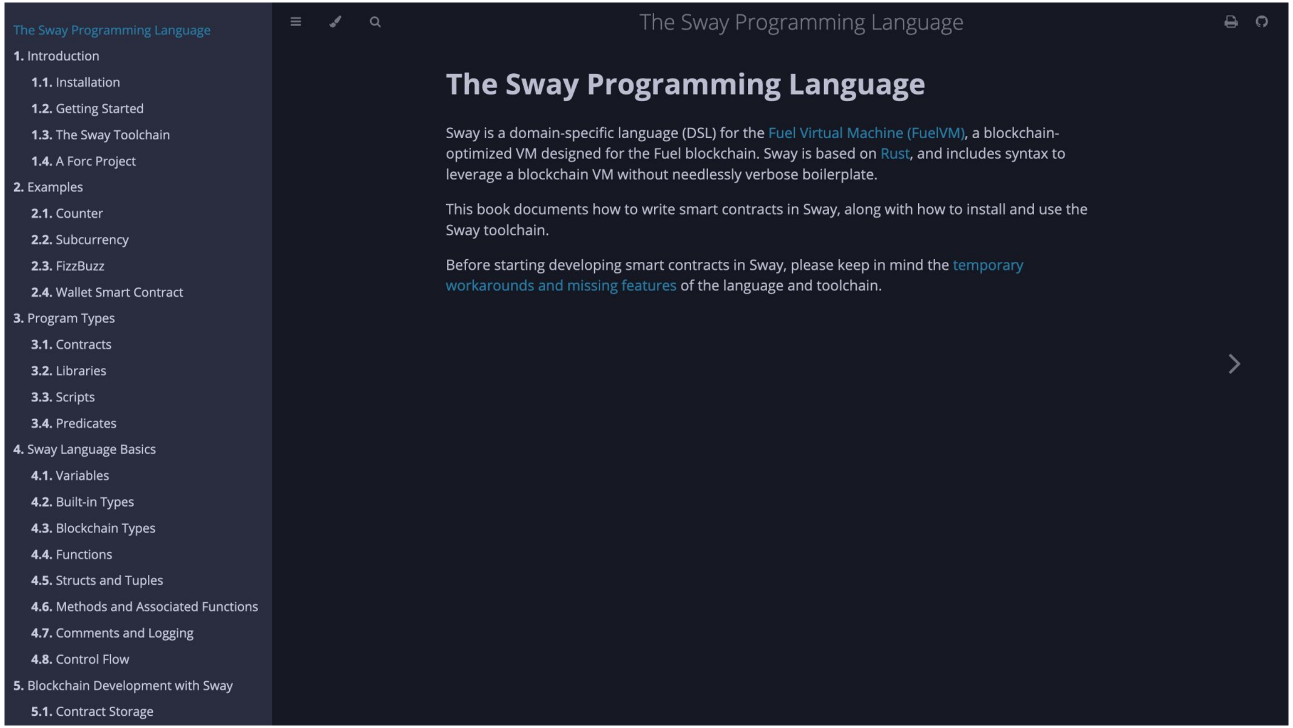
contract storage:



```
storage {
    data: u64
}

#[storage(read, write)]
fn simple_storage(amount: u64) -> u64 {
    storage.data = amount;
    storage.data
}
```

⚡ in active development ⚡

and much more…
fuel.network



The Sway Programming Language

The Sway Programming Language

Sway is a domain-specific language (DSL) for the Fuel Virtual Machine (FuelVM), a blockchain-optimized VM designed for the Fuel blockchain. Sway is based on Rust, and includes syntax to leverage a blockchain VM without needlessly verbose boilerplate.

This book documents how to write smart contracts in Sway, along with how to install and use the Sway toolchain.

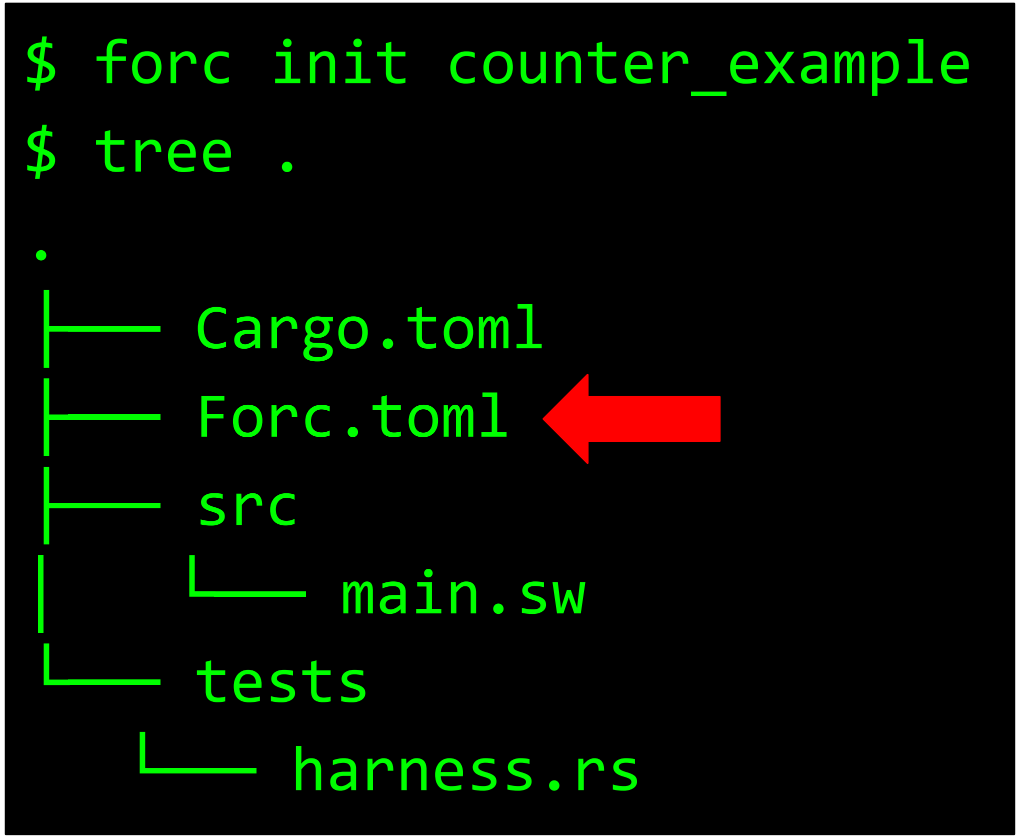
Before starting developing smart contracts in Sway, please keep in mind the temporary workarounds and missing features of the language and toolchain.

Building a “counter” example

install:

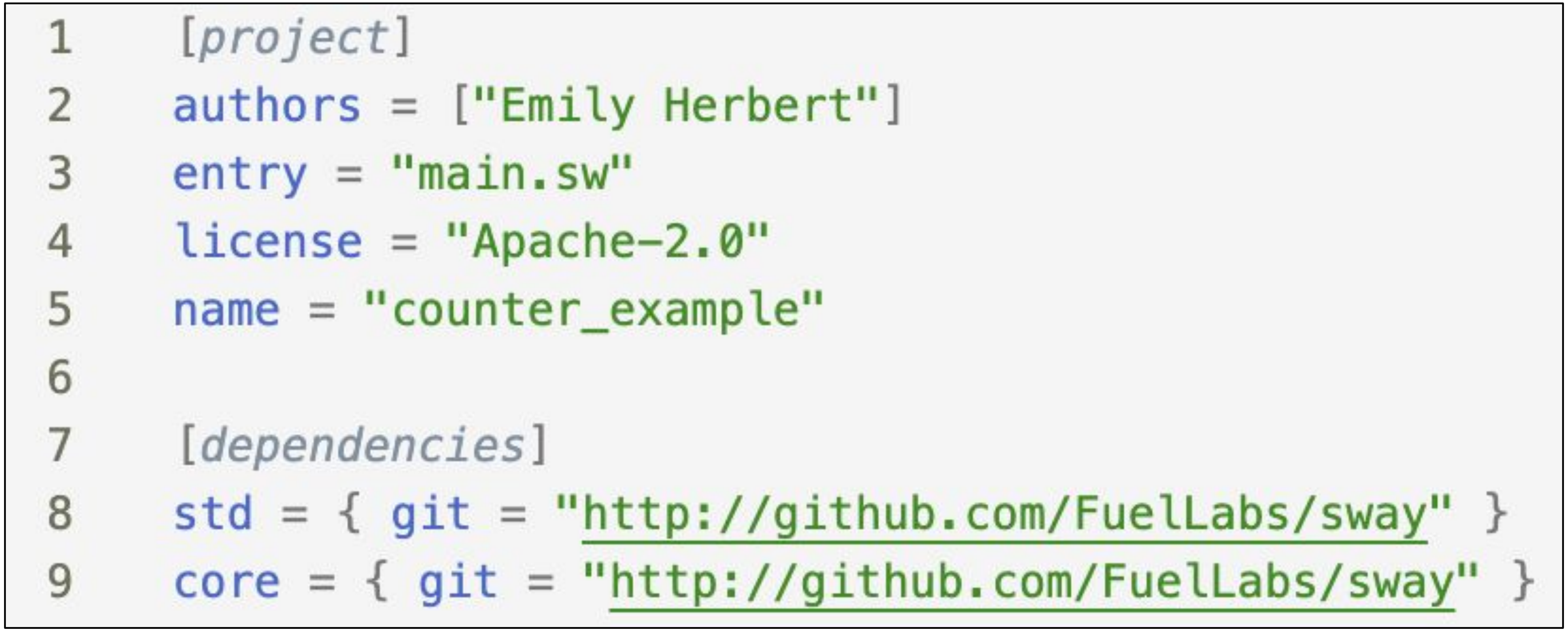
```
$ cargo install forc fuel-core
```

create a new Sway project:

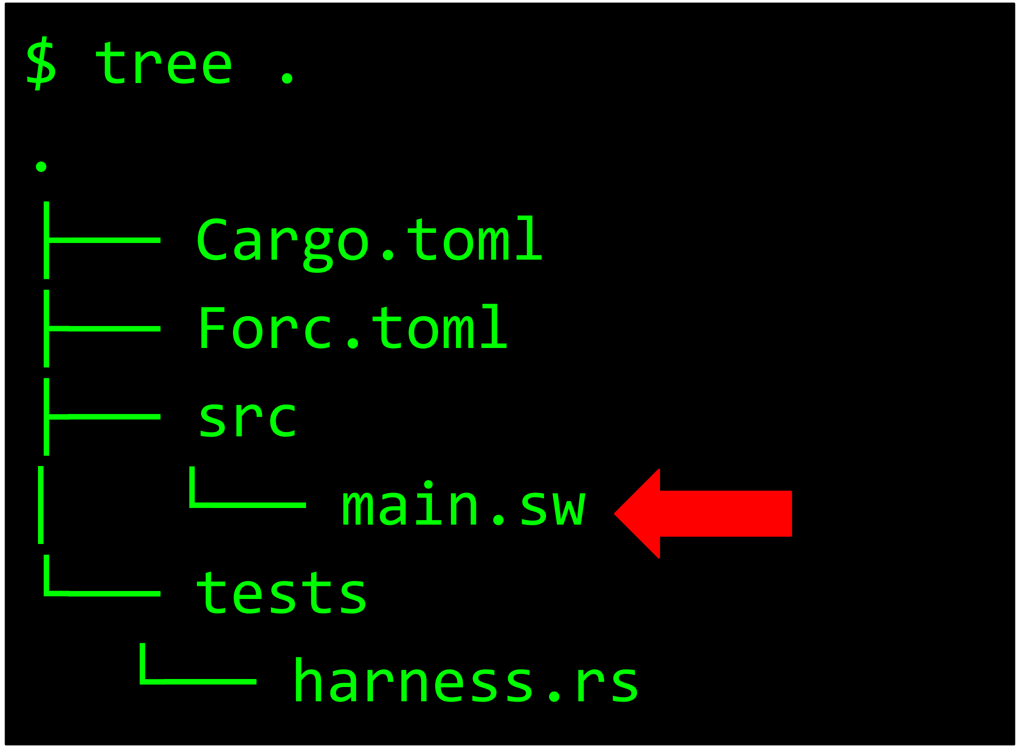


```
$ forc init counter_example
$ tree .
.
├── Cargo.toml
├── Forc.toml
├── src
│   └── main.sw
└── tests
    └── harness.rs
```

Forc.toml:

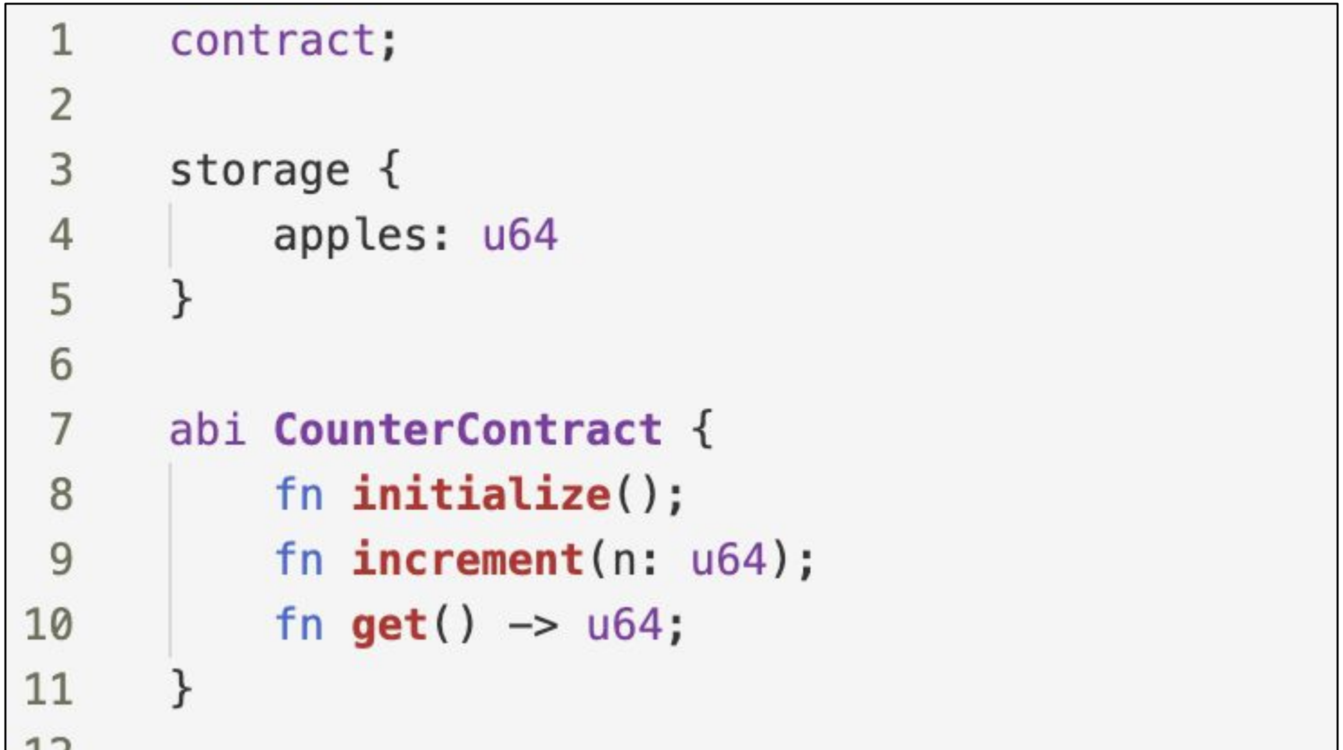


```toml
[project]
authors = ["Emily Herbert"]
entry = "main.sw"
license = "Apache-2.0"
name = "counter_example"

[dependencies]
std = { git = "http://github.com/FuelLabs/sway" }
core = { git = "http://github.com/FuelLabs/sway" }
```

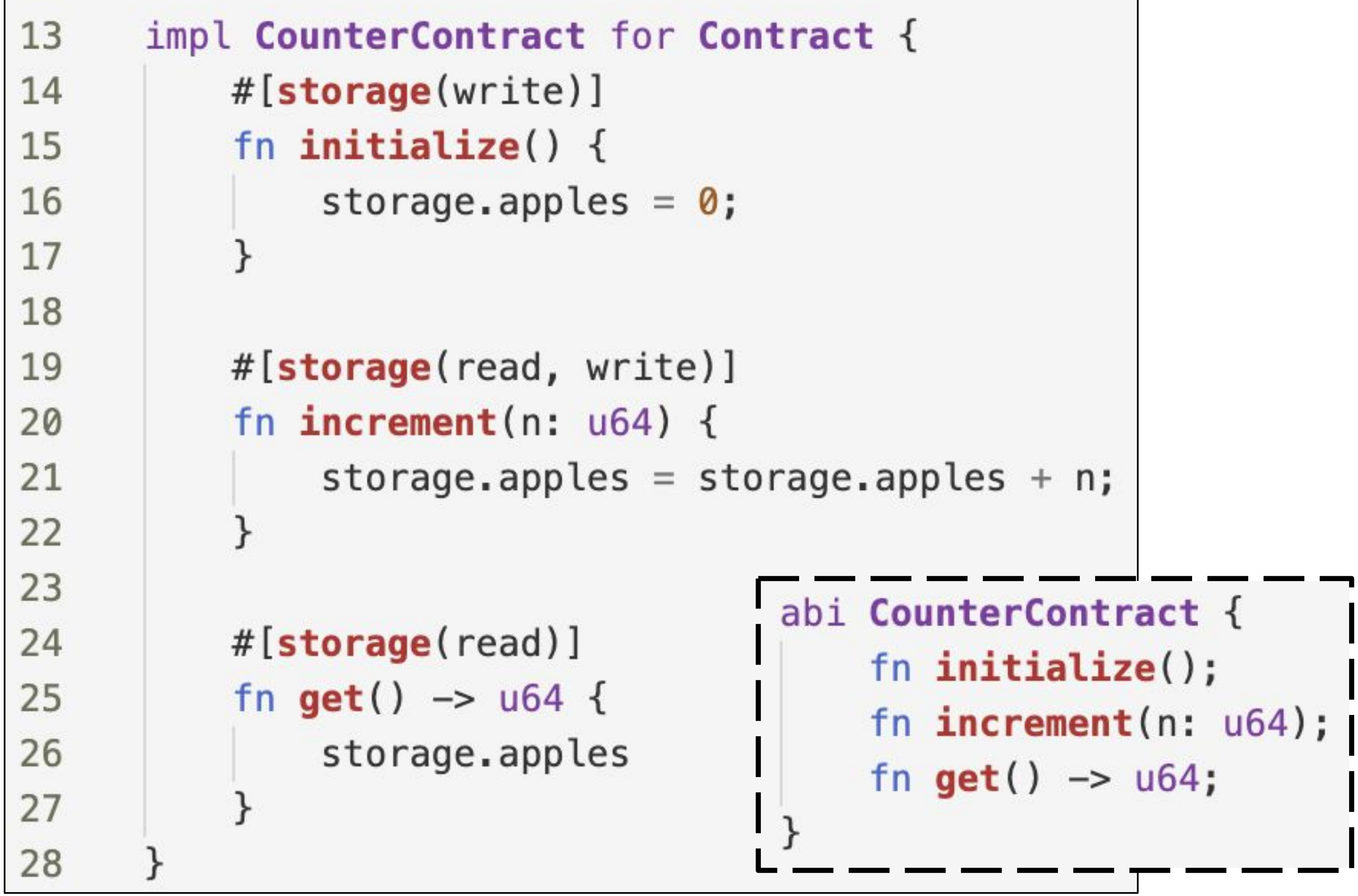


```
$ tree .

.
├── Cargo.toml
├── Forc.toml
├── src
│   └── main.sw
└── tests
    └── harness.rs
```

main.sw:

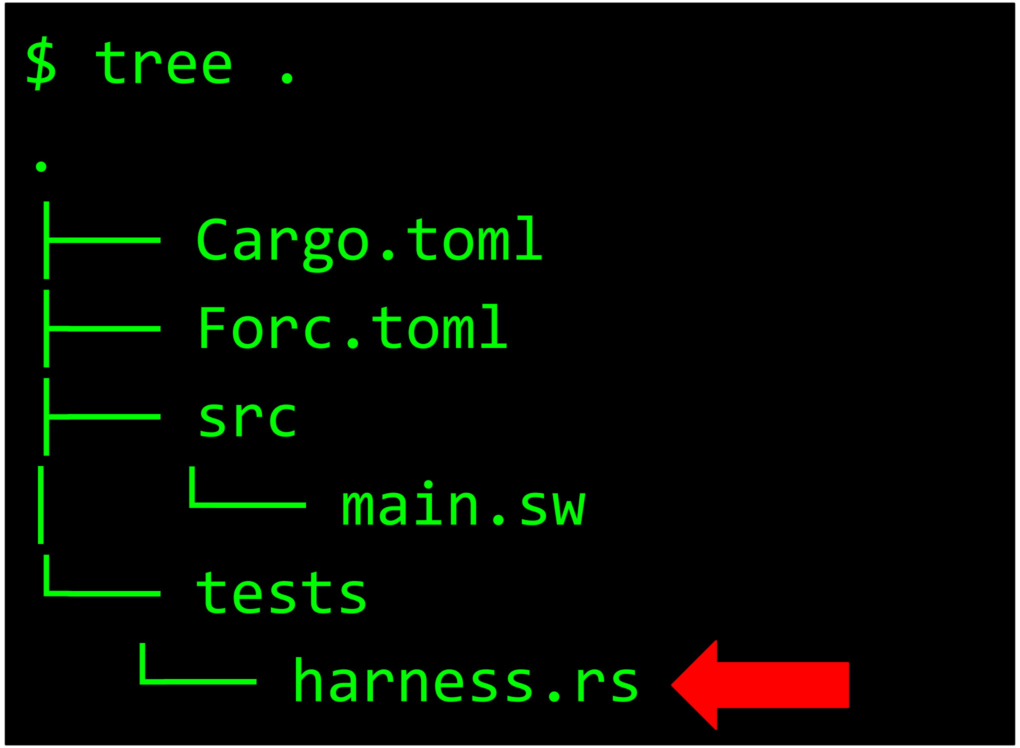


```
contract;

storage {
    apples: u64
}

abi CounterContract {
    fn initialize();
    fn increment(n: u64);
    fn get() -> u64;
}
```



```
impl CounterContract for Contract {
    #[storage(write)]
    fn initialize() {
        storage.apples = 0;
    }

    #[storage(read, write)]
    fn increment(n: u64) {
        storage.apples = storage.apples + n;
    }

    #[storage(read)]
    fn get() -> u64 {
        storage.apples
    }
}
```

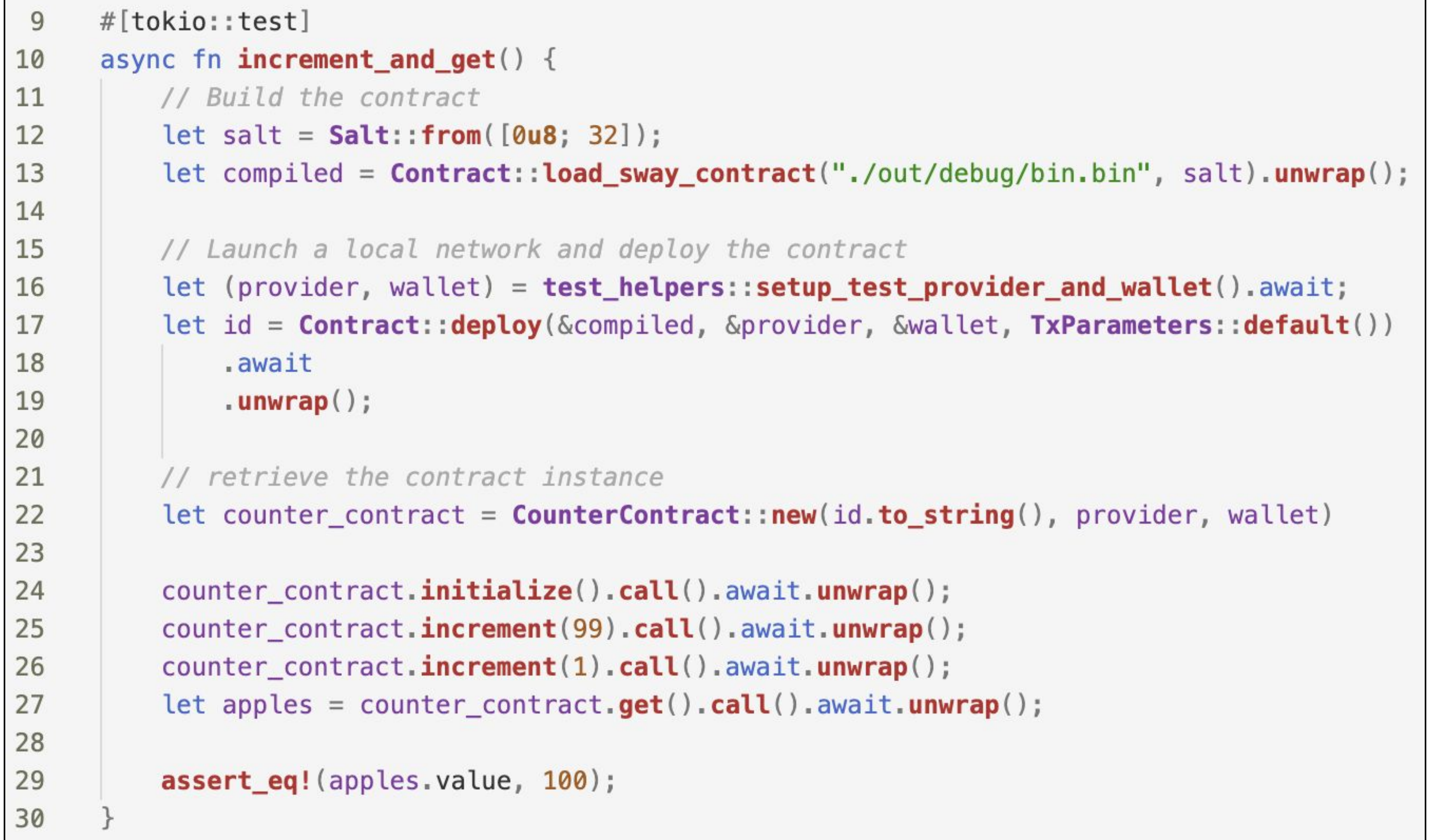
```
abi CounterContract {
    fn initialize();
    fn increment(n: u64);
    fn get() -> u64;
}
```



```
$ tree .

.
├── Cargo.toml
├── Forc.toml
├── src
│   └── main.sw
└── tests
    └── harness.rs
```

test harness:

```rust
use fuel_tx::Salt;
use fuels_abigen_macro::abigen;
use fuels_contract::{contract::Contract, parameters::TxParameters};
use fuels_signers::util::test_helpers;

// Load abi from json
abigen!(CounterContract, "out/debug/abi.json");

```



```rust
#[tokio::test]
async fn increment_and_get() {
    // Build the contract
    let salt = Salt::from([0u8; 32]);
    let compiled = Contract::load_sway_contract("./out/debug/bin.bin", salt).unwrap();

    // Launch a local network and deploy the contract
    let (provider, wallet) = test_helpers::setup_test_provider_and_wallet().await;
    let id = Contract::deploy(&compiled, &provider, &wallet, TxParameters::default())
        .await
        .unwrap();

    // retrieve the contract instance
    let counter_contract = CounterContract::new(id.to_string(), provider, wallet)

    counter_contract.initialize().call().await.unwrap();
    counter_contract.increment(99).call().await.unwrap();
    counter_contract.increment(1).call().await.unwrap();
    let apples = counter_contract.get().call().await.unwrap();

    assert_eq!(apples.value, 100);
}
```

run the test:

```
$ forc test
  Compiled library "core".
  Compiled library "std".
  Compiled contract "counter_example".
  Bytecode size is 300 bytes.
   Compiling counter_example v0.1.0

running 1 test
test increment_and_get ... ok
```

⭐ Done! ⭐

🤔 what if we make a mistake …. ? 🤔

Sway compiler can detect:

- programming errors
- blockchain-specific errors

$$\begin{tikzcd}\text{Sway} \arrow[d] \\ \text{FuelVM} \arrow[d] \\ \text{Fuel}\end{tikzcd}$$

the fastest modular execution layer

- Forc: the Sway toolchain
- friendly compiler
- built-in testing infra
- extensive SDK
- extensive standard library
- VSCode plugin
- language server (since May 2021)
- built-in debugger +
  gas profiling +
  code coverage (WIP)

ControlCplusControlV

Victor Lopez

Emily Herbert

Yuvraj Singh

Joshua Batty

Rodrigo Araújo

Oleksii Filonenko

Brandon Vrooman

Hannes Karppila

Brandon Kite

John Adler

Nicholas Furfaro

TJ Sharp

Toby Hutton

Elliot Yaghoobia

Austin O'Brien

Mohammad Fawaz

Ruben Amar

Samuel Aaron

Elvis Dedic

Michael Christenson II

Nick Alexander

Simon Roberts

Alex Hansen

Scott Williams

Dragan Rakita

Quinn Lee

Luiz Estacio

Mitchell Nordine

John Cub

Victor Nepveu

Andrew Cann

Thank you!

Fuel Labs: @fuellabs_

Sway: @SwayLang

me: @emilyaherbert