

Soroban Quest:

Introduction to Smart
Contracts & Soroban



Let's see who's here?



Who has used the blockchain before?

Maybe you've owned crypto, made a DApp, created an NFT.

Let's see who's here?



Who has written a smart contract before?

What are smart contracts?

Programs that run in the blockchain.

They can be as simple as “Hello World” or as complex as an entire application.

Soroban is a platform to write smart contracts on the **Stellar** blockchain, using the **Rust** programming language.



Stellar is a blockchain that's faster, cheaper, and far more energy-efficient than most blockchain-based systems.



Rust is a compiled programming language built for efficiency.



Why use smart contracts?

- Verifiable** Because contracts are kept on the blockchain, you can see what they do and be guaranteed the terms won't change.
- Fast** Contracts can be executed in near real-time, meaning you don't need a human to check their email or a bank to transfer funds.
- Direct** Because contracts are executed by code, it can remove the need for third parties (like bankers or Escrow companies) to verify terms and take action.
- On Chain** As you're already on a blockchain, it's easy to move tokens on that chain, even across borders.

Why use smart contracts?

Imagine being able to write a smart contract with your roommates that automatically splits rent each month and pays your landlord.

Verifiable

Because it's **verifiable** your roommates could see what they were agreeing to easily.

Fast

Because it's **fast** it would do it immediately when rent is due.

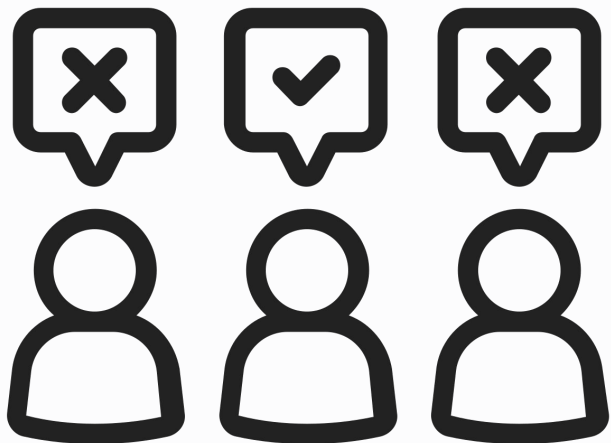
Direct

Because it's **direct** it would automatically work without you needing to involve your bank, setting up a shared account.

On Chain

Because it's **on chain** it has access to tokenized financial infrastructure easily, so even as a foreign exchange student it's easy to pay.

Why use smart contracts?



What are some other uses for smart contracts that you can think of?

Remember they are verifiable, fast, direct, and on chain.

Together, we'll write a smart contract.

Along with Blossom Bernice Breydenblach & Roscoe...

Getting Started



Go To:

<https://hackp.ac/sq>



Click

OPEN IN GITPOD >



Login or Create

a GitPod account

Select *VS Code (Browser)* as your editor.

Soroban

Discover the Golden Abacus and unlock the knowledge of the Soroban Quest.



Hello World

Bee and Roscoe set out on an exhilarating expedition to the enigmatic Squaktahune Temple, where they face a mysterious entrance with ancient symbols. But what could they mean?

[OPEN IN GITPOD >](#)



Log in to Gitpod



Continue with GitLab



Continue with GitHub



Continue with Bitbucket

Login to Soroban Quest

In the Terminal pane of Gitpod, type:

```
> sq user
```

If you get an error, try:

```
> sq login
```

You'll need a Discord account to login.

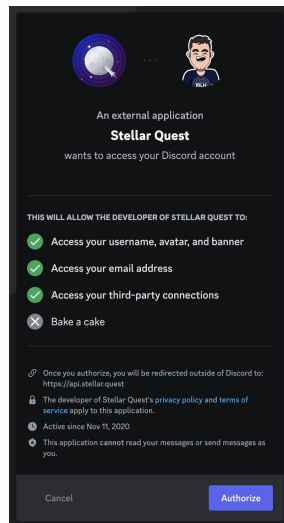


```
gitpod /workspace/soroban-quest (main) $ sq user
Please run the <login> command first
gitpod /workspace/soroban-quest (main) $ sq login
? Do you agree to abide by our Official Rules? > Yes
```

```
✓ Successfully authenticated
```

- ✗ Please connect your Stellar wallet
- ✗ Please complete the KYC flow
- ✗ Please upload your tax documents

```
? Your account is not yet fully complete.
This could affect your ability to claim either NFT or XLM rewards.
Would you like to complete your Stellar Quest account? (y/n) >
```



Questing Basics

Each quest has a directory with a README, explaining the entire quest.

We're working on the first quest.

If it's not already open, in the quests folder:



Open

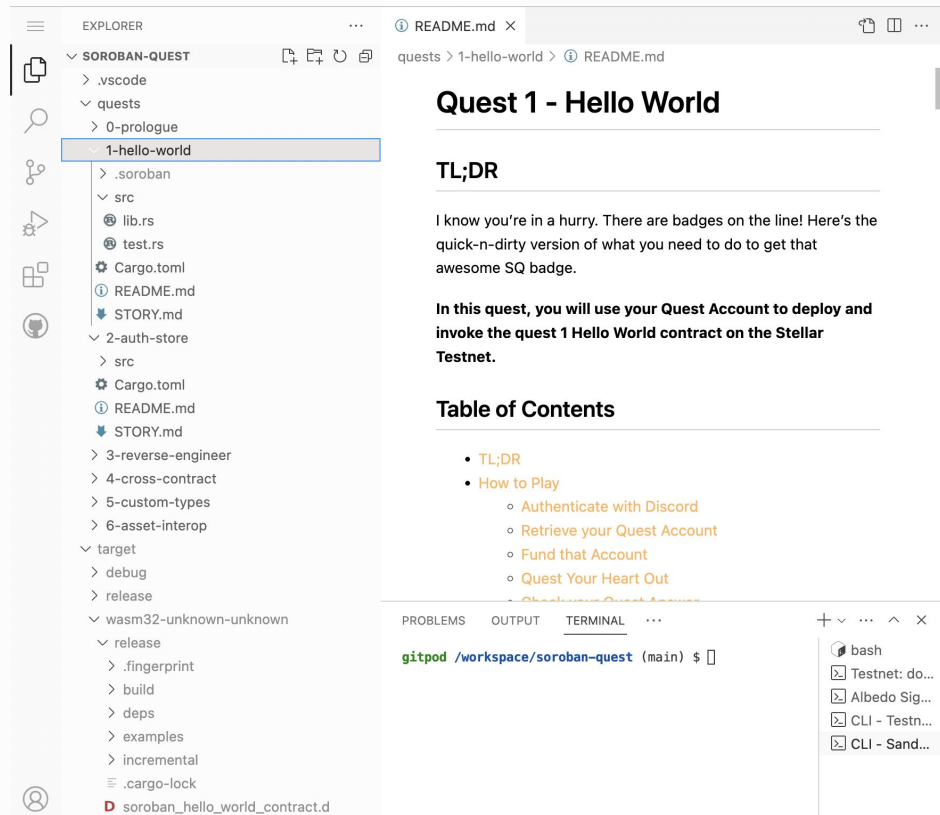
1-hello-world

To start a quest, you run a terminal command.

Start the first quest, in the Terminal, type:



sq play 1



The screenshot shows a VS Code workspace for a Soroban quest. The Explorer view on the left shows a directory structure under 'SOROBAN-QUEST' with subfolders for 'quests' and 'target', and various files like 'Cargo.toml', 'README.md', and 'STORY.md'. The '1-hello-world' quest directory is selected. The main editor shows the README for 'Quest 1 - Hello World', which includes a 'TL;DR' section, a paragraph of introductory text, and a 'Table of Contents' with links to 'TL;DR', 'How to Play', 'Authenticate with Discord', 'Retrieve your Quest Account', 'Fund that Account', and 'Quest Your Heart Out'. The Terminal view at the bottom shows the command prompt 'gitpod /workspace/soroban-quest (main) \$'.

Write a Contract

Soroban contracts are code written in Rust.

Soroban Quest 1 has a simple Hello World Contract.



Open
src/lib.rs

This contract has a function `hello()`.

The `hello()` function takes a short string and returns a *vector*:

```
["Hello", <string>]
```

```
11 // Our `HelloContract` implementation contains only one function, `hello()`.
12 // This function will receive a `to` argument, and return a Vec made up of
13 // "Hello" and the supplied `to` value.
14 #[contractimpl]
15 impl HelloContract {
16     pub fn hello(env: Env, to: Symbol) -> Vec<Symbol> {
17         // We use the `symbol_short` macro here, since our supplied string is
18         // fewer than 10 characters. For strings up to 32 characters, use
19         // `Symbol::new`.
20         vec![&env, symbol_short!("Hello"), to]
21     }
22 }
23
24 // This `mod` declaration inserts the contents of `test.rs` into this file.
25 mod test;
26
```



vector is a Rust datatype, similar to a Python List or a Javascript Array

A Little Housekeeping

Before we proceed, there is a minor edit we need to make to the Soroban Quest repository before moving forward.

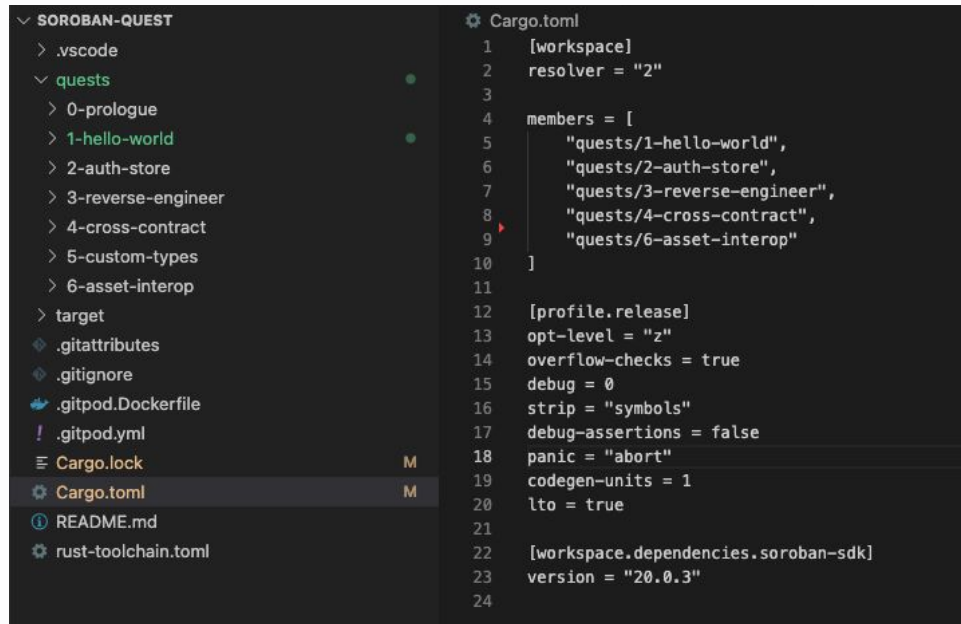


Open
Cargo.toml

Within that file, you'll see a *members* array.

Go ahead and delete

“quests/5-custom-types”, from the array.



```
▼ SOROBAN-QUEST
  > .vscode
  ▼ quests
    > 0-prologue
    > 1-hello-world
    > 2-auth-store
    > 3-reverse-engineer
    > 4-cross-contract
    > 5-custom-types
    > 6-asset-interop
  > target
  ◆ .gitattributes
  ◆ .gitignore
  🐳 .gitpod.Dockerfile
  ! .gitpod.yml
  ≡ Cargo.lock
  ⚙️ Cargo.toml
  ⓘ README.md
  ⚙️ rust-toolchain.toml

⚙️ Cargo.toml
1 [workspace]
2   resolver = "2"
3
4 members = [
5   "quests/1-hello-world",
6   "quests/2-auth-store",
7   "quests/3-reverse-engineer",
8   "quests/4-cross-contract",
9   "quests/6-asset-interop"
10 ]
11
12 [profile.release]
13   opt-level = "z"
14   overflow-checks = true
15   debug = 0
16   strip = "symbols"
17   debug-assertions = false
18   panic = "abort"
19   codegen-units = 1
20   lto = true
21
22 [workspace.dependencies.soroban-sdk]
23   version = "20.0.3"
24
```

We recently discovered a bug associated with the Quest 5 files in this repository, and while the Soroban team works to resolve that bug, this work around will allow us to continue to the next step free of errors.

Build a Contract

Rust is a compiled language, so in order to get this contract onto the Soroban blockchain we have to compile it into WebAssembly (*often simply called Wasm*).

To compile our contract, first move to the right directory:

```
> cd quests/1-hello-world
```

Now, run:

```
> soroban contract build
```

```
gitpod /workspace/soroban-quest (main) $ cd quests/1-hello-world/  
gitpod /workspace/soroban-quest/quests/1-hello-world (main) $ cargo build --release  
Finished release [optimized] target(s) in 0.08s
```


Deploying our Contract

So far, we have a contract that has been written, then compiled to Wasm, now we need to get it onto the Stellar blockchain.

The last command compiled our contract for a 32 bit WebAssembly architecture, the compiled file is at: `target/wasm32-unknown-unknown/release/soroban_hello_world_contract.wasm`

We need to deploy our contract to the Stellar testnet.

Run the deploy command for our file:

```
> soroban contract deploy --wasm  
../../target/wasm32-unknown-unknown/release/soroban_hello_world_contract.wasm
```

This will give you an alphanumeric
Contract Address.

```
gitpod /workspace/soroban-quest/quests/1-hello-world (main) $ soroban contract deploy  
--wasm ../../target/wasm32-unknown-unknown/release/soroba  
n_hello_world_contract.wasm  
CDB3YVE47TYFFSMNVAMWLZGLDS3D75EVMHGBH640DWDWTN4L2VHYSQMQ
```

Invoking our Contract

Once deployed, all that's left is to run the code in our contract!

Invoke the contract, running the hello function:

```
> soroban contract invoke --id <contract_addr> -- hello --to "You"
```

```
gitpod /workspace/soroban-quest/quests/1-hello-world (main) $ soroban contract invoke  
--id CBV4YwECBBJKYTP7QBR0TAZ30YYJJLEOXWYMADA5A42I5E4TWYHTLRZ6 -- hello --to "You"  
["Hello","You"]
```

Understanding our Invocation



What happened when we ran our command?

```
> soroban contract invoke --id <contract_addr> -- hello --to "You"
```

soroban contract invoke This command creates an interface to call smart contract functions.

--id <contract_addr> Specifies the address of the contract.
You should substitute <contract_addr> with the address of your contract that you got from the soroban contract deploy command.

-- This passes any future input to the contract you've invoked.

hello The name of the function you want to run.

--to The name of the argument from your Rust function.

"You" The string you're passing in the to argument.
You can change this to anything you want as long as it's less than 32 characters.

Completing the Quest

You did it! You just deployed a Smart Contract and ran it on the blockchain!

Great work!

Completing a Soroban Quest earns you NFTs, let's complete it!



> *sq check 1*

What's next?

Keep learning and building on Soroban...

Now that you've deployed your first contract on Soroban, you can keep going to build larger contracts and full applications.

1. **Keep Building on Soroban Quest**

Soroban Quest will lead you through more core concepts on Soroban, you can try it yourself, by going on to 2-auth-store or you can follow walkthroughs like what we just did by going to: <https://hackp.ac/sorobanquest-helloworld>

2. **Read the Soroban Docs**

Soroban has great docs that will help you understand Soroban Quest and how to use Soroban: <https://hackp.ac/soroban-docs>

3. **Better Understand Rust**

Soroban uses Rust to create Smart Contracts, to get great at creating them, it's helpful to know how the language works, you can checkout some great resources:

Rust by Example: <https://doc.rust-lang.org/rust-by-example/>

A Gentle Introduction to Rust: <https://stevedonovan.github.io/rust-gentle-intro/>