



Compile contract locally and run it

Imitating the blockchain environment to test

You've done it. You've written a smart contract. You're a champ!

Now we need to actually

1. Compile it.
2. Deploy it to our local blockchain.
3. Once it's there, that `console.log` will run :).

We need to do this because in the real world, smart contracts live on the blockchain. And, we want our website and smart contract to be used by real people so they can 🙌 at us or do whatever you want them to do!

So, even when we're working locally we want to imitate that environment. Technically, we could just compile and run Solidity code, but what makes Solidity magical is how it can interact with the blockchain and Ethereum wallets (which we'll see more of in the next lesson). So, better to just knock this out right now.

We're just going to write a custom script that handles those 3 steps for us.

Let's do it!

Build a script to run our contract

Go into the `scripts` directory and make a file named `run.js`.

So, to test a smart contract we've got to do a bunch of stuff right. Like: compile, deploy, then execute.

Our script will make it really easy to iterate on our contract really fast :).

So, this is what `run.js` is going to have:

```
const main = async () => {  
  const waveContractFactory = await hre.ethers.getContractFactory("WavePortal");  
  const waveContract = await waveContractFactory.deploy();  
  await waveContract.deployed();  
}
```

[Previous](#)

[Submit Terminal Output Screenshot](#)

```
const runMain = async () => {
  try {
    await main();
    process.exit(0); // exit Node process without error
  } catch (error) {
    console.log(error);
    process.exit(1); // exit Node process while indicating 'Uncaught Fatal Exception' error
  }
  // Read more about Node exit ('process.exit(num)') status codes here: https://stackoverflow
};

runMain();
```

Awesome.

🤔 How's it work?

Again going line by line here.

```
const waveContractFactory = await hre.ethers.getContractFactory("WavePortal");
```

This will actually compile our contract and generate the necessary files we need to work with our contract under the `artifacts` directory. Go check it out after you run this :).

```
const waveContract = await waveContractFactory.deploy();
```

This is pretty fancy :).

What's happening here is Hardhat will create a local Ethereum network for us, but just for this contract. Then, after the script completes it'll destroy that local network. So, every time you run the contract, it'll be a fresh blockchain. What's the point? It's kinda like refreshing your local server every time so you always start from a clean slate which makes it easy to debug errors.

```
await waveContract.deployed();
```

Previous

Submit Terminal Output Screenshot

We'll wait until our contract is officially deployed to our local blockchain! Our `constructor` runs when we actually deploy.

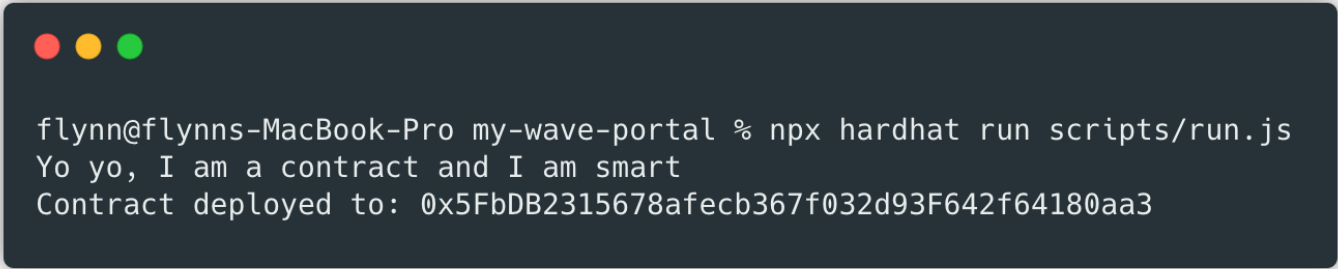
```
console.log("Contract deployed to:", waveContract.address);
```

Finally, once it's deployed `waveContract.address` will basically give us the address of the deployed contract. This address is how we can actually find our contract on the blockchain. There are millions of contracts on the actual blockchain. So, this address gives us easy access to the contract we're interested in working with! This will be more important a bit later once we deploy to a real Ethereum network.

Let's run it!

```
npx hardhat run scripts/run.js
```

You should see your `console.log` run from within the contract and then you should also see the contract address print out!!! Here's what I get:



```
flynn@flynns-MacBook-Pro my-wave-portal % npx hardhat run scripts/run.js
Yo yo, I am a contract and I am smart
Contract deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
```



Hardhat & HRE

In these code blocks you will constantly notice that we use `hre.ethers`, but `hre` is never imported anywhere? What type of magic trick is this?

Directly from the Hardhat docs themselves you will notice this:

The Hardhat Runtime Environment, or HRE for short, is an object containing all the functionality that Hardhat exposes when running a task, test or script. In reality, Hardhat is the HRE.

So what does this mean? Well, every time you run a terminal command that starts with `npx hardhat` you

[Previous](#)

[Submit Terminal Output Screenshot](#)

```
const hre = require("hardhat")
```

TL;DR - you will be seeing `hre` a lot in our code, but never imported anywhere! Checkout this cool [Hardhat documentation](#) to learn more about it!

Before you click "Next Lesson"

Note: if you don't do this, Farza will be very sad :(.

Go to #progress and post a screenshot of your terminal with the output.

Be sure to make that `console.log` whatever you want! You've now written your own contract and ran it by deploying to a local blockchain WOOOOOOOOOOO LETS GOOO.