I've put together a JavaScript library that can help people generate STARK-based proofs of computation. The goal is to take care of as much boilerplate code as possible, and make creating new STARKs simple and "easy."

The library is in this GitHub repo and it is also published on NPM. It is largely based on Vitalik Buterin's zk-STARK/MiMC tutorial - but it is highly generalized. For example, defining a MiMC STARK takes just 10 lines of code:

```
const mimcStark = new Stark({ field: new PrimeField(2n ** 256n - 351n * 2n ** 32n + 1n), tFunction: 'out: $r0^3 + $k0', tConstraints: '$n0 - ($r0^3 + $k0)', tConstraintDegree: 3, constants: [{ values : roundConstants, pattern : 'repeat' }] });
```

Defining a STARK to prove Fibonacci computation is only 11 lines:

```
const fibStark = new Stark({ field: new PrimeField(2n**32n - 3n * 2n**25n + 1n), tFunction: a0: $r0 + $r1; out: [a0, a0 + $r1];, tConstraints: a0: $r0 + $r1; out: [$n0 - a0, $n1 - (a0 + $r1)];, tConstraintDegree: 1 });
```

(here, we need to set up 2 registers because the framework is limited to 2 consecutive states, but Fibonacci sequences requires 3 consecutive states to validate).

Once you've defined a STARK, you can use it to make proofs and verify computations like so:

```
const inputs = [3n]; const steps = 2**13; const result = 952247743554997679519680487145663165977852976959036972351304343631225554760 56n; const assertions = [ { step: 0, register: 0, value: inputs[0] }, // value at first step is equal to input { step: steps - 1, register: 0, value: result } // value at last step is equal to result ];

let proof = mimcStark.prove(assertions, steps, inputs); // create a proof let result = mimcStark.verify(assertions, proof, steps); // verify the proof console.log(result); // true
```

The project is in its infancy right now, and there are still many things to fix and optimize (see the issues in the repo). So, would appreciate any feedback, help, and support.