gist:
Break a JavaScript jail that only allows these special characters: +*-

steps:
1. run script 2nd time runs smoother already
2. analyze limitations
3. research similar concepts
4. get the flag
5. proof and further ideas

1. In this case you just need to navigate to the folder of the challenge and run it via:

```
npm init -y
npm install express
node chall.js
```

2. Looking at the code, this jail is even more restrictive than the previous one:

```
const allowedChars = new Set('+*-![]()'.split(''));
for (const char of code) {
  if (!allowedChars.has(char)) {
    return res.status(400).send('Disallowed characters in code');
  }
}
```

So we can only use: +, *, -, !, [,], (, and)
Not even numbers or letters! At first glance, this looks impossible.

3. But the name of the challenge - **"wtf"** - reminded me of esoteric programming languages like **Brainfuck**. A quick search for "javascript only symbols programming" led me to **JSFuck**, and suddenly everything clicked!

JSFuck is an esoteric programming style that writes any JavaScript using only six characters: [], !, +, (, and). Our jail gives us these exact characters (plus * and -).

The core idea of JSFuck is that you can build up numbers and then strings using just these symbols:
- [] is an empty array
- +[] converts it to 0
- ![] gives false
- !![] or ! + []gives true

Wait, **true** is usually evaluated as **1**. Can we do **arithmetic** to get **numbers** that would be interpreted as **chars** and eventually to a **string**?

I played around with some input:

```
curl -X POST -H "Content-Type: application/json" -d '{"code":"![]"}' http://localhost:3000/eval
false

curl -X POST -H "Content-Type: application/json" -d '{"code":"!+[]"}' http://localhost:3000/eval
true
```

and also confirmed the limitations:

curl -X POST -H "Content-Type: application/json" -d '{"code":"this"}' http://localhost:3000/eval
Disallowed characters in code

then began to get daring!

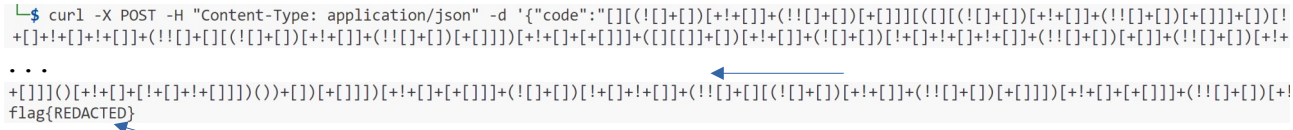
curl -X POST -H "Content-Type: application/json" -d '{"code":"!+[]+!+[]"}' http://localhost:3000/eval
2

This proofed exactly what I needed. Or as **Bertrand Russel** put it in the Principia Mathematica - "The above proposition is occasionally useful" (p.379) But creating a real string with that is tedious. There must be a tool. Of course, there is.

4. Finding a JSFuck converter online https://jsfuck.com/ and converting our previous solution:

Object.values(this)

The converter gave me back a MASSIVE string of symbols (seriously, it's huge).



And there it was! The **flag** appeared just like in the previous challenge, but this time through a much more sophisticated yet basic approach.

5. Further ideas:
This absolutely reminded me of my time as logics tutor at university. It really exploits ideas from the Russell-Whitehead type theory and the obviously automatic **type casting** and detection helps a lot.

While this worked perfectly, I wonder about:
- Whether there are shorter ways to achieve the same result and if it would even be worthwhile to optimise payloads
- If I could have manually constructed a smaller payload for just this specific case

Lessons learned:
1. Challenge names often contain vital hints - **talking names**
2. Knowledge of esoteric programming languages can be surprisingly useful
3. JavaScript's type coercion system is wild
4. Rusell Rules!

The fact that people not only discovered this was possible but turned it into a whole programming style is both impressive and slightly concerning.

Quod erat demonstrandum.