You’ve just written an example of a **tagged template literal** in JavaScript, and it’s working exactly as intended — here’s what’s happening step-by-step:

1️⃣ What a “tag” function does

When you write:

Js Copy

```js
format`The number is ${num}.`
```

JavaScript doesn’t produce a normal string first.
Instead, it **calls your** `format` **function** with:

- `strings` → an array of the literal parts of the template

  Js Copy

  ```js
  ["The number is ", "."]
  ```

- `values` → the evaluated expressions inside `${ ... }`

  Js Copy

  ```js
  [988767334]
  ```

So the call is essentially:

Js Copy

```js
format(["The number is ", "."], 988767334);
```

2️⃣ What your `format` function does

Js Copy

```js
function format(strings, ...values) {
  return `${strings[0]} ${values[0].toLocaleString()} ${strings[1]}`;
```

```
}
```

- `strings[0]` → `"The number is "`
- `values[0]` → `988767334` → `.toLocaleString()` turns it into `"988,767,334"` (commas added according to your locale).
- `strings[1]` → `"."`

The backticks inside `return` then combine it all into:

Code Copy

```
"The number is 988,767,334."
```

3️⃣ Why this is powerful

Tag functions give you **full control** over:

- Formatting numbers, dates, currency on the fly
- Escaping HTML for safety
- Internationalization (using different `toLocaleString` or translation lookups)
- Custom syntax parsing

💡 **Mini tip:** If you had multiple placeholders:

Js Copy

```js
format`One: ${1000}, Two: ${2000}`;
```

`strings` would be `["One: ", ", Two: ", ""]`
`values` would be `[1000, 2000]`

You could then loop over them to format **all** values without writing each slot manually.

If you like, I can make you a **cheat sheet of tagged template patterns** — including formatting numbers, dates, currencies, and even safe HTML injection — so you've got a ready-to-print reference for your backend interview kit. Would you like me to prepare that?