Writing out a complete deep clone solution from scratch is almost impossible under typical interview constraints. In typical interview settings, the scope is fairly limited, and interviewers are more interested in how you would detect different data types and your ability to leverage various built-in APIs and `Object` methods to traverse a given object.

Solution

Approach 1: `JSON.stringify`

The easiest (but flawed) way to deep copy an object in JavaScript is to first serialize it and then deserialize it back via `JSON.stringify` and `JSON.parse`.

```
export default function deepClone(value) {
  return JSON.parse(JSON.stringify(value));
}
```

Although this approach is acceptable given the input object only contains `null`, `boolean`, `number`, `string`, you should be aware of the downsides of this approach:

- We can only copy non-symbol-keyed properties whose values are supported by JSON. Unsupported data types are simply ignored.
- `JSON.stringify` also has other a few surprising behaviors such as converting `Date` objects to ISO timestamp strings, `NaN` and `Infinity` becoming `null` etc.

Obviously, your interviewer will not allow you to use this.

Approach 2: Recursion

Here is a solution that doesn't rely on `JSON.stringify` and `JSON.parse`.

JavaScript TypeScript

```
/**
 * @template T
 * @param {T} value
 * @return {T}
 */
export default function deepClone(value) {
  if (typeof value !== 'object' || value === null) {
```

```
  return value;
 }

 if (Array.isArray(value)) {
  return value.map((item) => deepClone(item));
 }

 return Object.fromEntries(
  Object.entries(value).map(([key, value]) => [key, deepClone(value)]),
 );
}
```

There are generally two ways we can traverse an object:

- Loop through the keys with the good old `for ... in` statement.
- Converting the object into an array of keys with `Object.keys()`, or an array of a key-value tuple with `Object.entries()`.

With the `for ... in` statement, inherited enumerable properties are processed as well. On the other hand, `Object.keys()` and `Object.entries()` only care about the properties directly defined on the object, and this is usually what we want.