{Object} Cloning in JavaScript

SWIPE →

Munib Shaikh

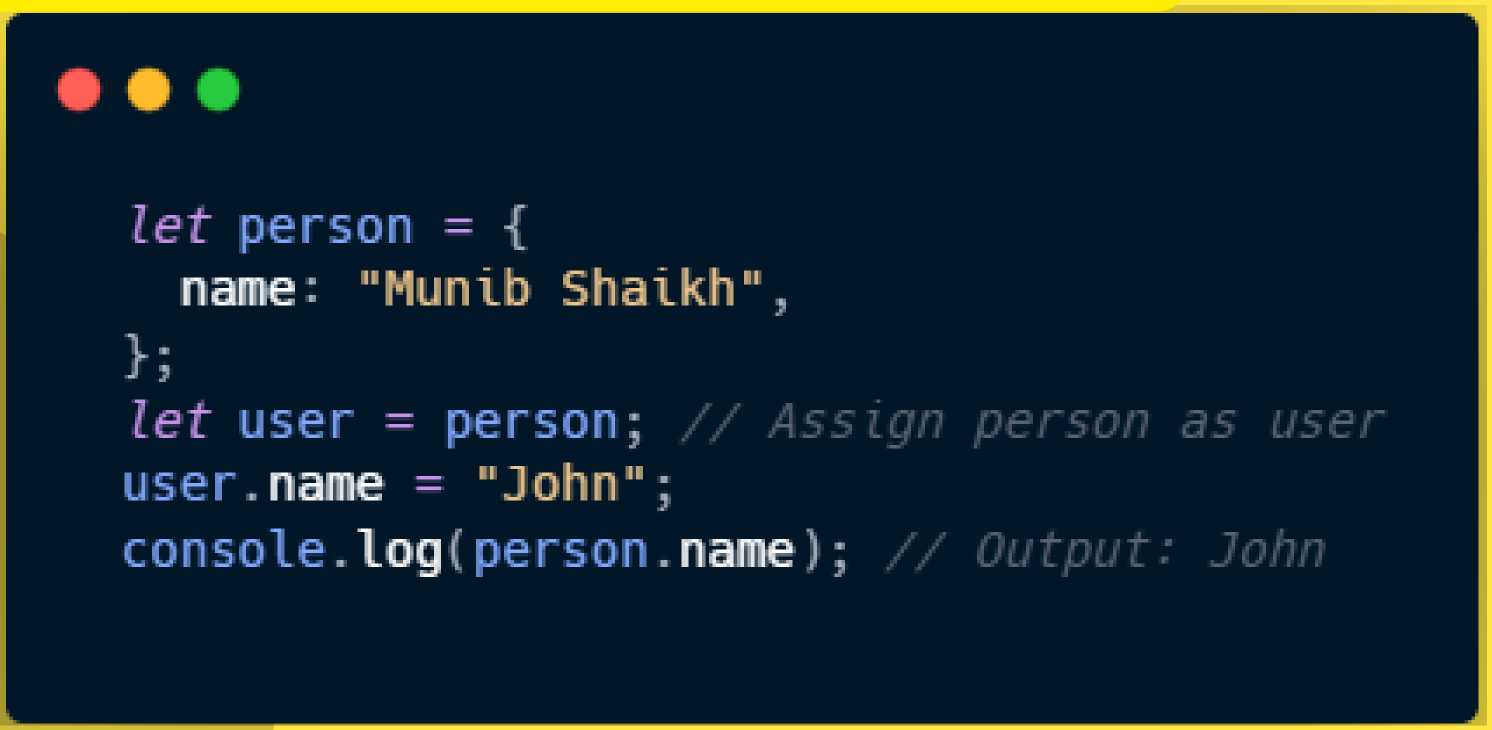
WHY?

(DEEP & SHALLOW COPY)

In javascript, we have two concepts of copy:

- **Copy by Value (Data)**
- **Copy by Reference**

when we assign one object to another, it copies that object's reference (Memory Location), not the actual data. That's why we use deep and shallow copies to keep the original value safe.

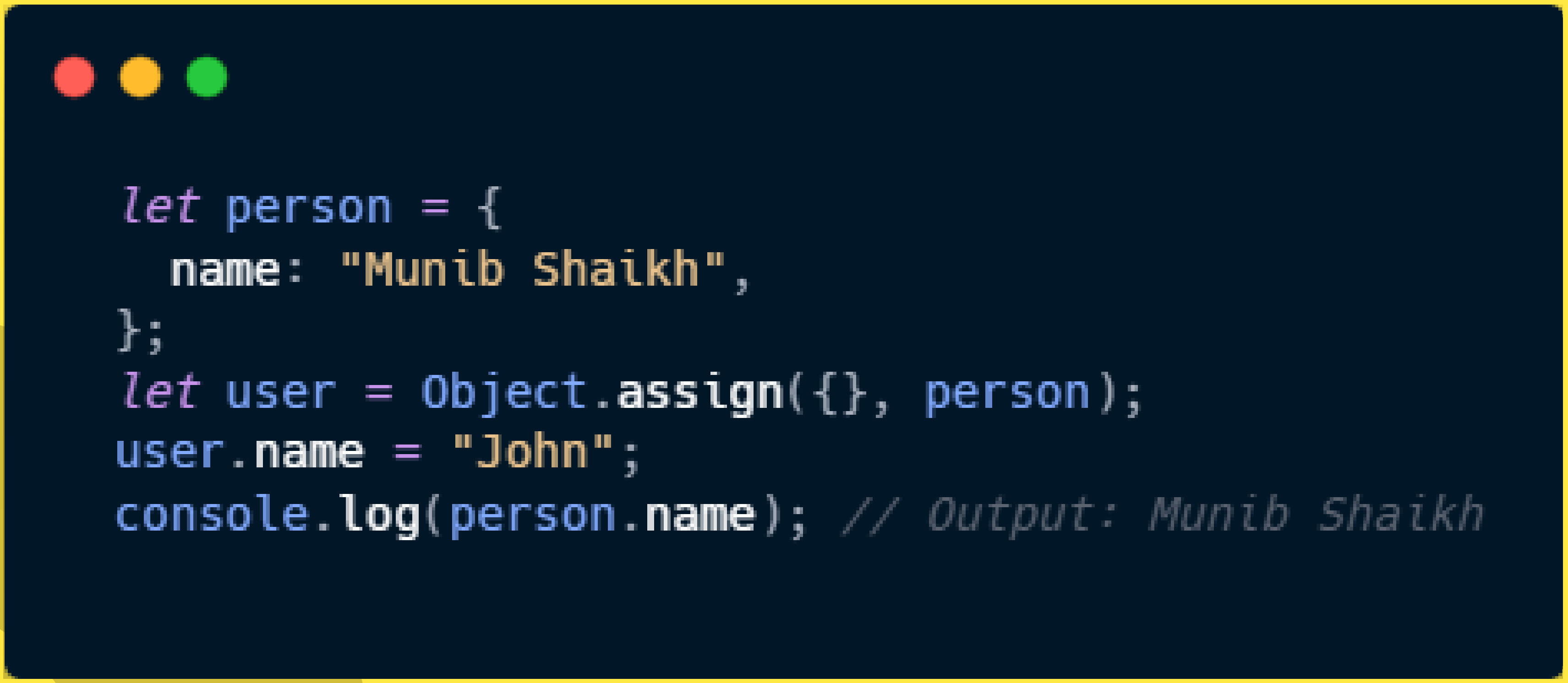


```
let person = {
  name: "Munib Shaikh",
};
let user = person; // Assign person as user
user.name = "John";
console.log(person.name); // Output: John
```

SHALLOW COPY

METHOD: 01 (Object.assign)

- **Object.assign(): it copies the object by value, not by reference. which means the value that exists in the main/original object does not change.**
- **Object.assign method copy all the properties from one or more objects to a targeted object.**

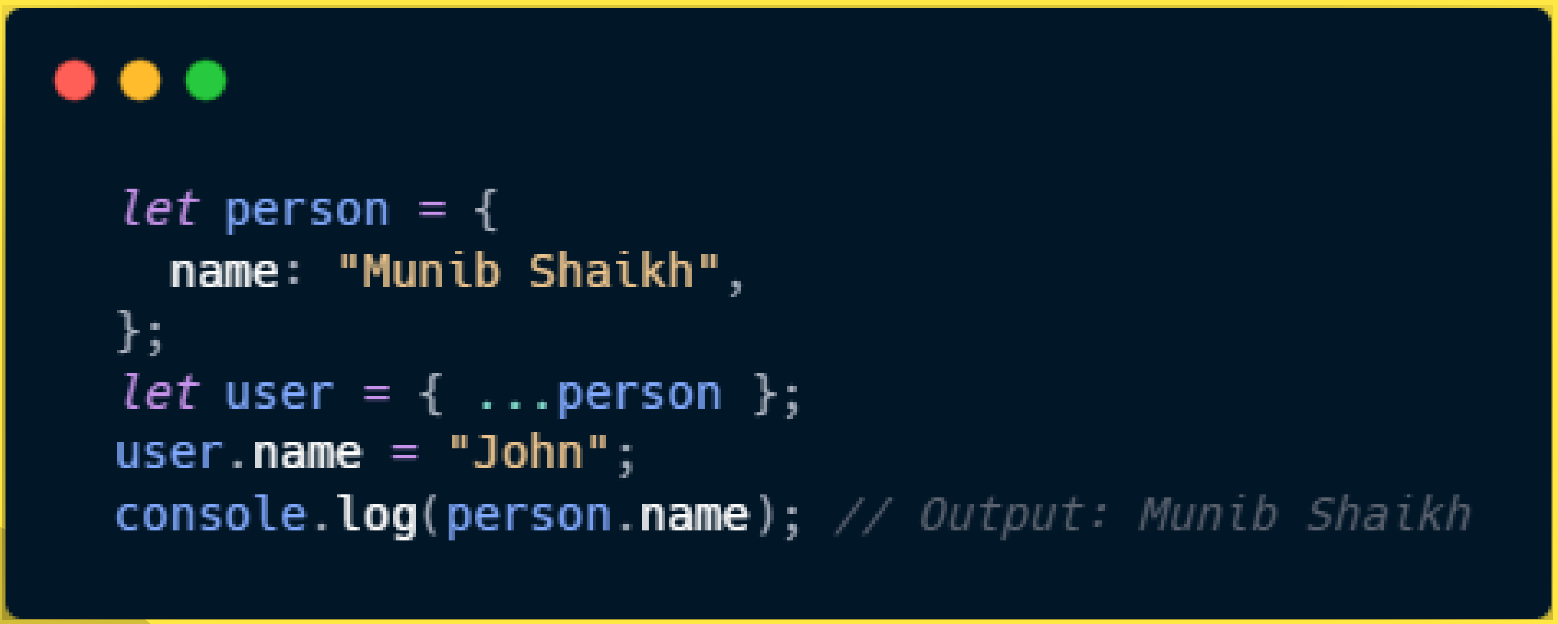


```
let person = {
  name: "Munib Shaikh",
};
let user = Object.assign({}, person);
user.name = "John";
console.log(person.name); // Output: Munib Shaikh
```

Munib Shaikh

METHOD: 02 (DESTRUCTURING)

Spread Operator (...): It also copies the object by value, not by reference. The spread operator was introduced in ECMAScript 6. The spread operator can make code more pretty and concise.

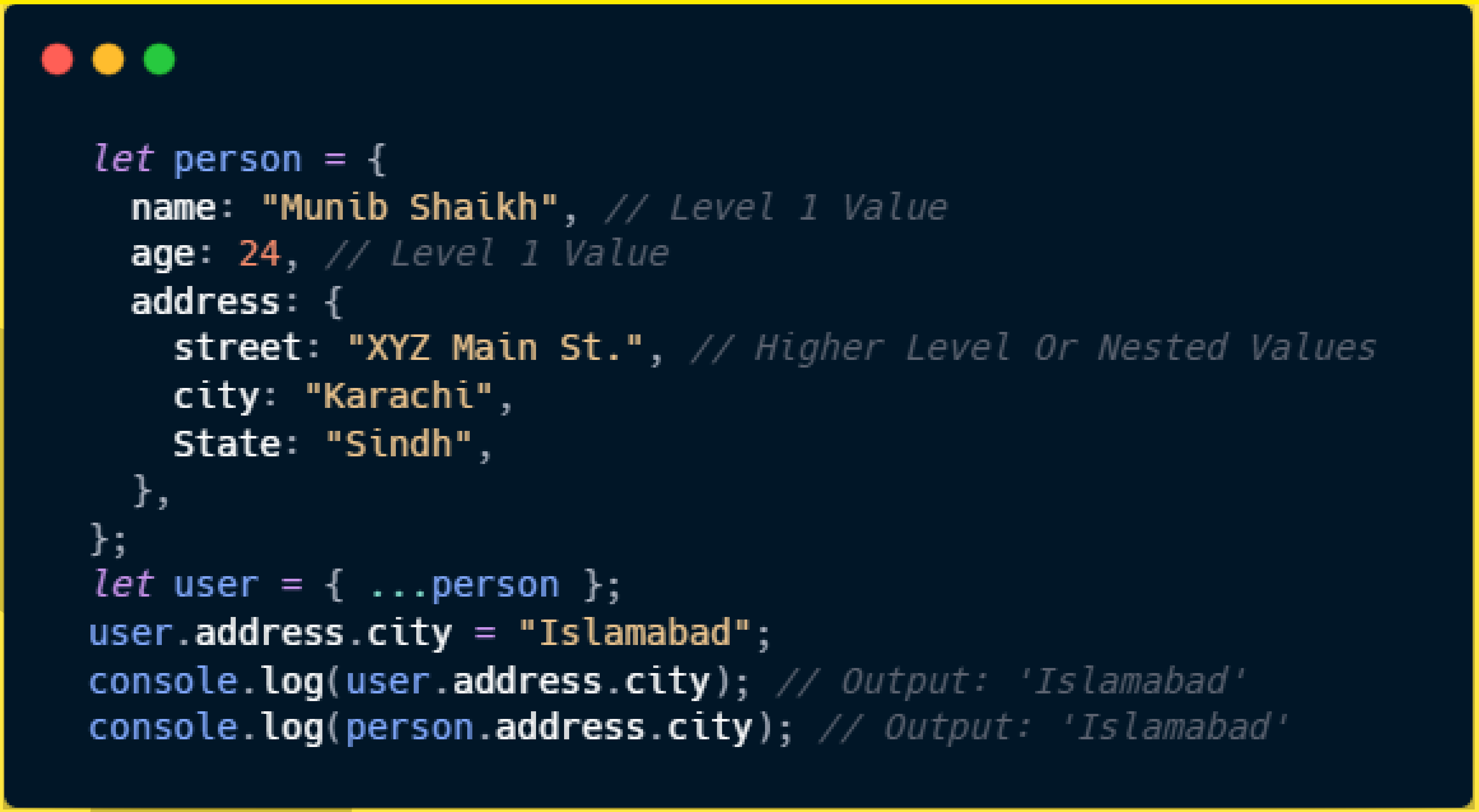


```
let person = {
  name: "Munib Shaikh",
};
let user = { ...person };
user.name = "John";
console.log(person.name); // Output: Munib Shaikh
```

Munib Shaikh

WHY DEEP COPY?

Shallow Copy copies only the top-level values of an object using data, not reference. For nested or higher-level values, it retains references. This means changes to nested values in a cloned object affect the original object's nested values. Deep Copy, however, duplicates nested values using data, ensuring independence between the cloned and original object's nested values.



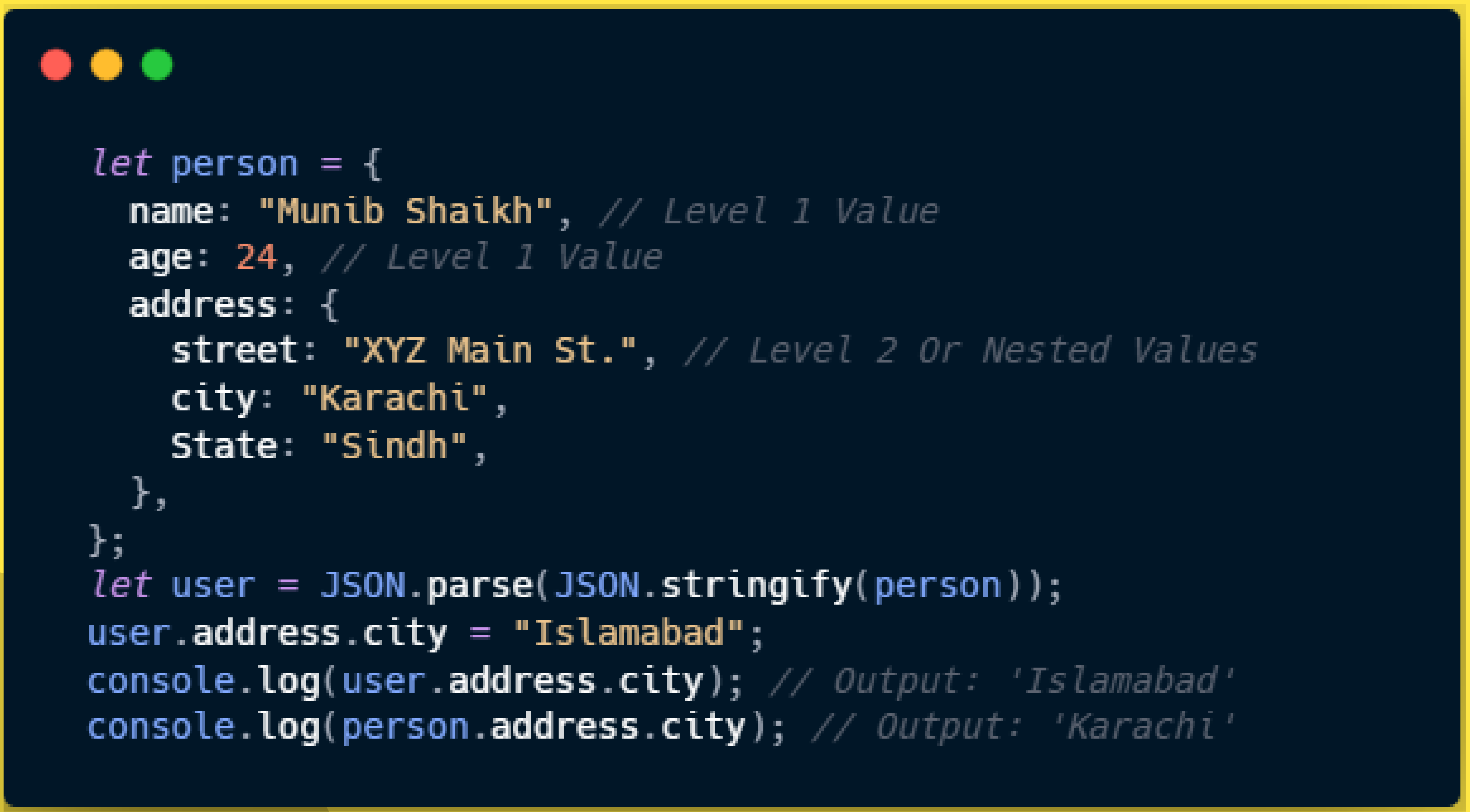
```
let person = {
  name: "Munib Shaikh", // Level 1 Value
  age: 24, // Level 1 Value
  address: {
    street: "XYZ Main St.", // Higher Level Or Nested Values
    city: "Karachi",
    State: "Sindh",
  },
};
let user = { ...person };
user.address.city = "Islamabad";
console.log(user.address.city); // Output: 'Islamabad'
console.log(person.address.city); // Output: 'Islamabad'
```

Munib Shaikh

DEEP COPY

METHOD: 01 (Stringify)

JSON's methods (stringify, parse) helps to achieve a deep copy of the object

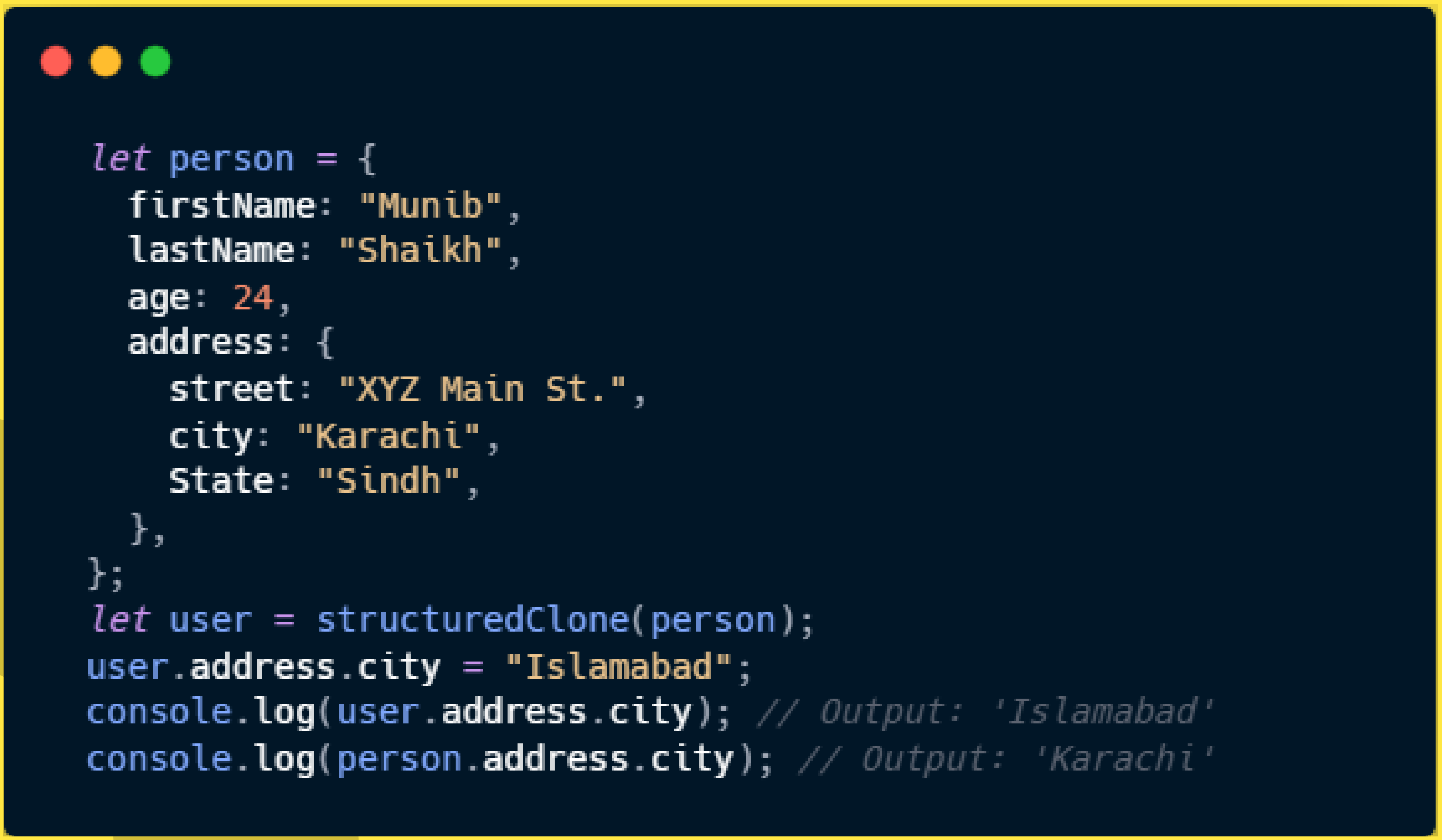


```
let person = {
  name: "Munib Shaikh", // Level 1 Value
  age: 24, // Level 1 Value
  address: {
    street: "XYZ Main St.", // Level 2 Or Nested Values
    city: "Karachi",
    State: "Sindh",
  },
};
let user = JSON.parse(JSON.stringify(person));
user.address.city = "Islamabad";
console.log(user.address.city); // Output: 'Islamabad'
console.log(person.address.city); // Output: 'Karachi'
```

Munib Shaikh

METHOD: 02 (structuredClone):

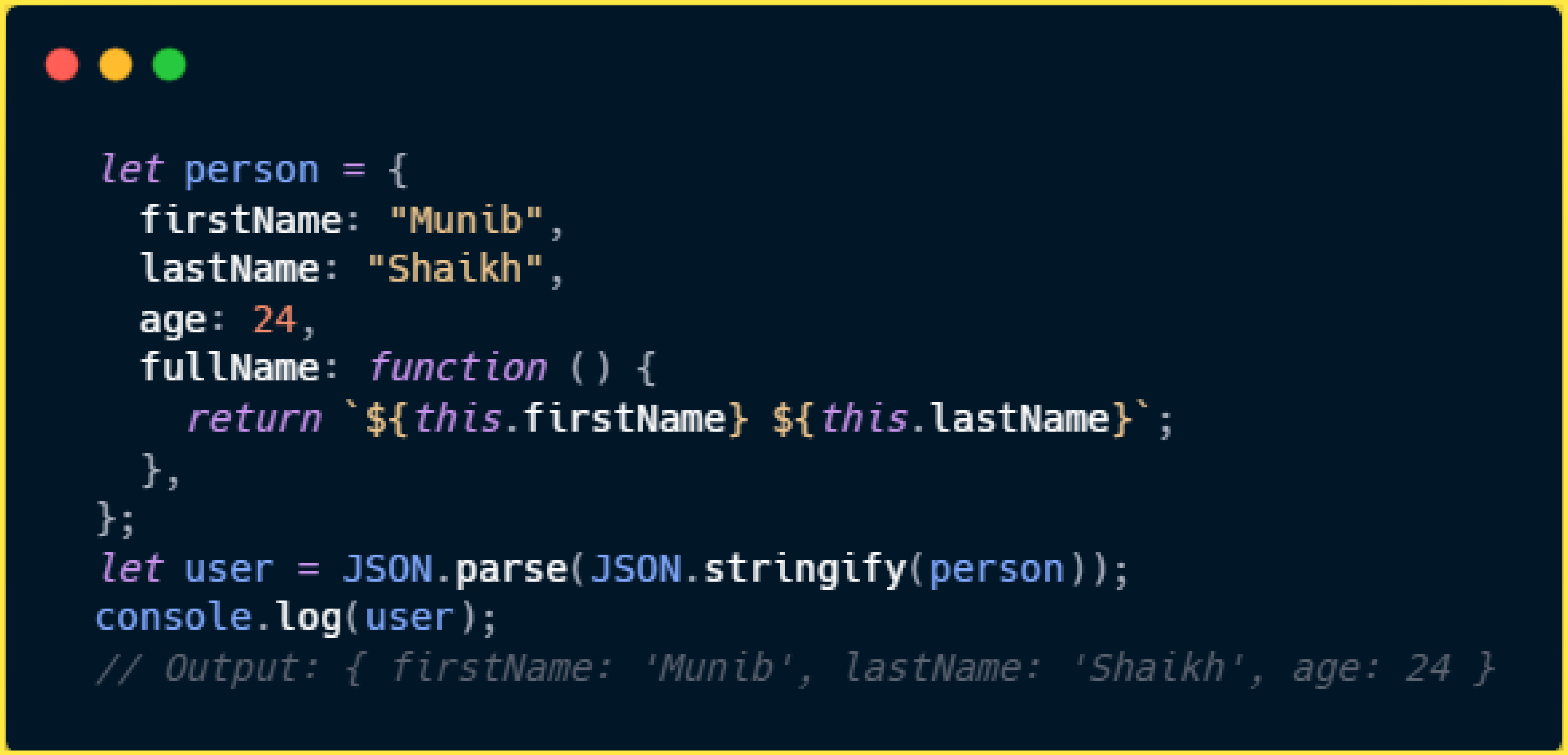
The structuredClone method in JavaScript allows us to create deep copies of complex objects, like arrays and nested objects. This is helpful when we need to share data between different parts of a web application that are running at the same time, like in web workers.



```
let person = {
  firstName: "Munib",
  lastName: "Shaikh",
  age: 24,
  address: {
    street: "XYZ Main St.",
    city: "Karachi",
    State: "Sindh",
  },
};
let user = structuredClone(person);
user.address.city = "Islamabad";
console.log(user.address.city); // Output: 'Islamabad'
console.log(person.address.city); // Output: 'Karachi'
```

LIMITATIONS:

Make sure these methods have limitations.



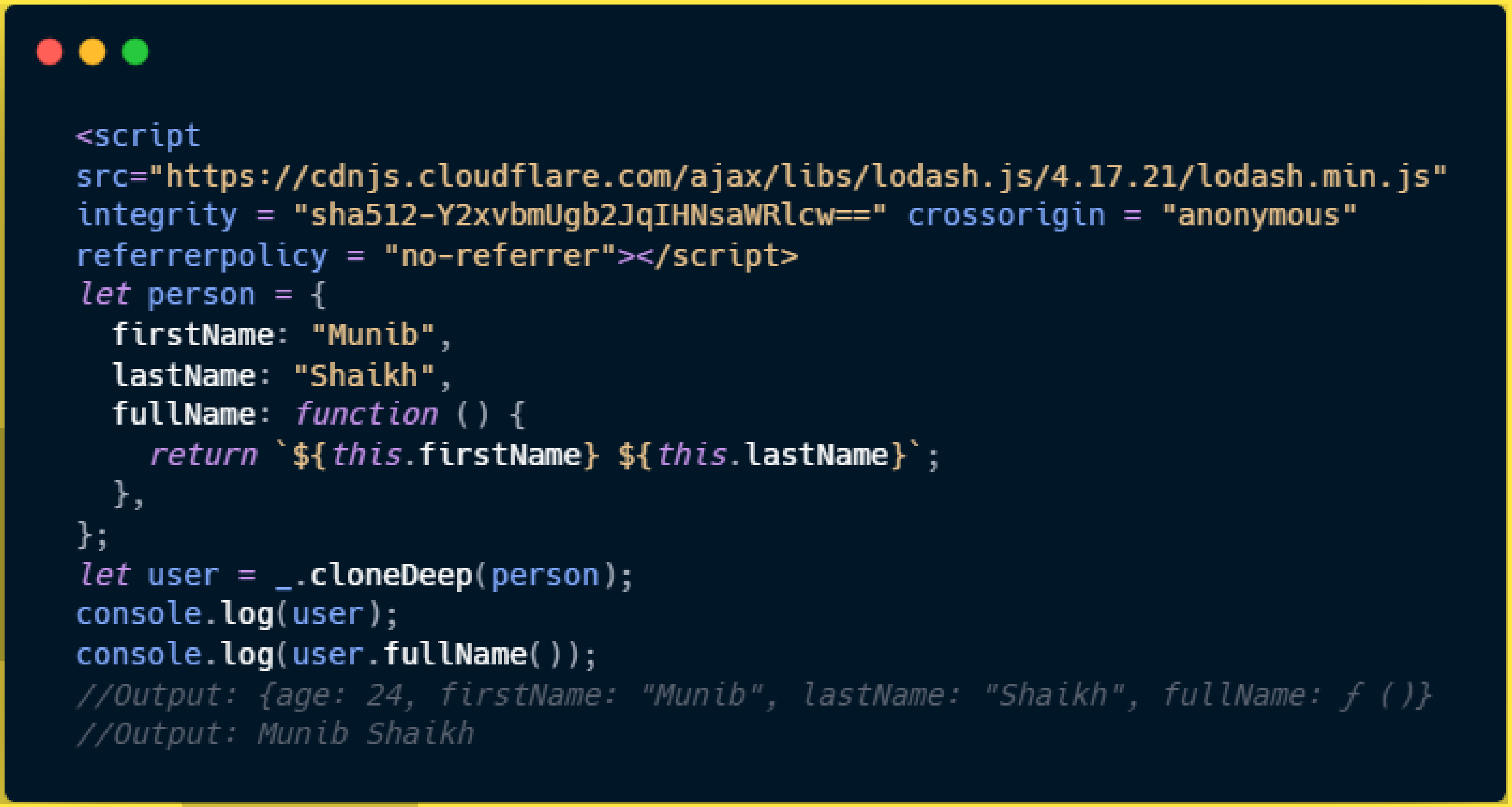
```javascript
let person = {
  firstName: "Munib",
  lastName: "Shaikh",
  age: 24,
  fullName: function () {
    return `${this.firstName} ${this.lastName}`;
  },
};
let user = JSON.parse(JSON.stringify(person));
console.log(user);
// Output: { firstName: 'Munib', lastName: 'Shaikh', age: 24 }
```

It cannot copy the function inside the object.

Munib Shaikh

METHOD: 03 (LODASH) OVERCOMING CONSTRAINTS

Loadash is a library that helps to copy the function and values which exists in the object. CDN and node modules both are available so don't worry if you use a framework/library or even simple JS.



```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.21/lodash.min.js"
integrity = "sha512-Y2xvbmUgb2JqIHNsaWRlcw==" crossorigin = "anonymous"
referrerpolicy = "no-referrer"></script>
let person = {
  firstName: "Munib",
  lastName: "Shaikh",
  fullName: function () {
    return `${this.firstName} ${this.lastName}`;
  },
};
let user = _.cloneDeep(person);
console.log(user);
console.log(user.fullName());
//Output: {age: 24, firstName: "Munib", lastName: "Shaikh", fullName: ƒ ()}
//Output: Munib Shaikh
```

The choice between deep copy and shallow copy depends on your project's needs. Deep copy is the best option if you want complete separation and independence. If efficiency and shared references are a priority, then a shallow copy can be more suitable. By understanding the differences between these methods, you can choose the one that meets your programming needs.

Munib Shaikh

I hope this helps you a little. feel free to reach out if you have any questions or feedback.

COMMENT BELOW

Munib Shaikh