1.4 Javascript Objects and Modules

Table of Content

1. Javascript Objects

JavaScript object is a non-primitive data type that allows you to store multiple collections of data. If you are familiar with other programming languages, JavaScript objects are a bit different. You do not need to create classes in order to create objects.

```
// object
const student = {
    firstName: 'ram',
    class: 10
};
```

You can also create values with nested objects, arrays, functions, etc.

▼ **Accessing Object Properties**

1. **Using dot Notation**

```
const person = {
    name: 'John',
    age: 20,
};

// accessing property
console.log(person.name); // John
```

2. **Using bracket Notation**

```
const person = {
    name: 'John',
    age: 20,
};

// accessing property
console.log(person["name"]); // John
```

▼ **Javascript Object Methods**

You can create functions as values inside an object.

```
// object containing method
const person = {
    name: 'John',
    greet: function() { console.log('hello'); }
};
```

Javascript `this` keyword

To access a property of an object from within a method of the same object, you need to use the `this` keyword. Let's consider an example.

```
const person = {
    name: 'John',
    age: 30,

    // accessing name property by using this.name
    greet: function() { console.log('The name is' + ' ' +
};

person.greet();
```

2. Javascript Modules

As our program grows bigger, it may contain many lines of code. Instead of putting everything in a single file, you can use modules to separate codes in separate files according to their functions. This makes our code organized and easier to maintain.

Benefits of Using Modules:

- The code base is easier to maintain because different codes having different functionalities are in different files.
- Makes code reusable. You can define a module and use it numerous times as per your needs.

▼ **export and import**

```
// exporting a function
export function greetPerson(name) {
    return `Hello ${name}`;
}
```

```
// importing greetPerson from greet.js file
import { greetPerson } from './greet.js';

// using greetPerson() defined in greet.js
let displayName = greetPerson('Jack');
```

▼ **Rename export and import**

```
export {
    function1 as newName1,
    function2 as newName2
};
```

```
import { function1 as newName1, function2 as newName2 } fr
```

▼ **Default export**

```
// default export
export default function greet(name) {
    return `Hello ${name}`;
}
```

```
export const age = 23;
```

```
import random_name from './greet.js';
```

While performing default export,

- `random_name` is imported from `greet.js`. Since, `random_name` is not in `greet.js`, the default export (`greet()` in this case) is exported as `random_name`.
- You can directly use the default export without enclosing curly brackets `{}`.

> 💡 **Note**: A file can contain multiple exports. However, you can only have one default export in a file.

Assignments

- Create a JavaScript object with properties `name`, `age`, and a method `greet` that prints a greeting.
- Write an ES6 module exporting a function `add` and import it into another file.
- Combine two arrays into one using the spread operator.