Array of Objects

An array of objects is a collection of objects where each object represents an individual data item with key-value pairs. Understanding how to manually work with arrays of objects (without built-in array functions like map, filter, reduce, etc.) is crucial for mastering fundamental programming logic.

1. Accessing Properties

To access properties, loop through the array using a for or while loop.

Example:
Print the name of each student.

```
let students = [
  { name: "Alice", age: 20, marks: 85 },
  { name: "Bob", age: 22, marks: 90 },
  { name: "Charlie", age: 21, marks: 78 }
];

for (let i = 0; i < students.length; i++) {
  console.log(students[i].name);
}
// Output:
// Alice
// Bob
// Charlie
```

Explanation:

- The for loop iterates over each element in the array of students.
- students[i] refers to the object at the current index i.
- .name accesses the name property of the current object.
- This prints the name of each student one by one.

2. Adding a New Object

Manually add an object to the array using push.

Example:

Add a new student to the array.

```
let students = [
  { name: "Alice", age: 20, marks: 85 },
  { name: "Bob", age: 22, marks: 90 },
  { name: "Charlie", age: 21, marks: 78 }
];

students[students.length] = { name: "David", age: 23, marks: 88 };
console.log(students);

// Output:
// [
//   { name: 'Alice', age: 20, marks: 85 },
//   { name: 'Bob', age: 22, marks: 90 },
//   { name: 'Charlie', age: 21, marks: 78 },
//   { name: 'David', age: 23, marks: 88 }
// ]
```

Explanation:

- `students.length` gives the current size of the array.
- Assigning a new object at index `students.length` effectively appends it to the array.
- A new object `{ name: "David", age: 23, marks: 88 }` is added without using push.

3. Modifying Object Properties

Access the object directly by its index and update its properties.

Example:

Change "Charlie's" marks to 80.

```
let students = [
  { name: "Alice", age: 20, marks: 85 },
  { name: "Bob", age: 22, marks: 90 },
```

```
  { name: "Charlie", age: 21, marks: 78 },
  { name: "David", age: 23, marks: 88 }
];


for (let i = 0; i < students.length; i++) {
  if (students[i].name === "Charlie") {
    students[i].marks = 80;
  }
}
console.log(students);
// Output:
// [
//   { name: 'Alice', age: 20, marks: 85 },
//   { name: 'Bob', age: 22, marks: 90 },
//   { name: 'Charlie', age: 21, marks: 80 },
//   { name: 'David', age: 23, marks: 88 }
// ]
```

Explanation:

- The loop iterates through the array to find the object where the name property matches "Charlie".
- When found, the marks property of that object is updated to 80.
- This demonstrates how to locate and modify specific data within an array of objects.

4. Searching for an Object

Manually iterate through the array to find an object with a specific property value.

Example:
Find the student named "Bob."

```
let students = [
  { name: "Alice", age: 20, marks: 85 },
  { name: "Bob", age: 22, marks: 90 },
  { name: "Charlie", age: 21, marks: 78 },
  { name: "David", age: 23, marks: 88 }
];
```

```
let foundStudent = null;
for (let i = 0; i < students.length; i++) {
  if (students[i].name === "Bob") {
    foundStudent = students[i];
    break;
  }
}
console.log(foundStudent);
// Output: { name: 'Bob', age: 22, marks: 90 }
```

Explanation:

- A for loop is used to search for a student with the name "Bob".
- If the condition matches, the object is assigned to foundStudent.
- break exits the loop early after finding the object, making the search more efficient.

5. Removing an Object

Manually remove an object by recreating the array without the unwanted object.

Example:
Remove the student named "Alice."

```
let students = [
  { name: "Alice", age: 20, marks: 85 },
  { name: "Bob", age: 22, marks: 90 },
  { name: "Charlie", age: 21, marks: 78 },
  { name: "David", age: 23, marks: 88 }
];
let updatedStudents = [];
for (let i = 0; i < students.length; i++) {
  if (students[i].name !== "Alice") {
    updatedStudents[updatedStudents.length] = students[i];
  }
}
console.log(updatedStudents);
// Output:
// [
//   { name: 'Bob', age: 22, marks: 90 },
//   { name: 'Charlie', age: 21, marks: 80 },
```

```
//   { name: 'David', age: 23, marks: 88 }
// ]
```

Explanation:

- A new array updatedStudents is created to store only the desired objects.
- The loop checks if the name property of each object is not "Alice".
- If the condition is true, the object is added to the new array, effectively removing "Alice".

6. Filtering Objects

Manually create a new array containing only objects that meet a condition.

Example:
Get students with marks greater than 80.

```
let highScorers = [];
for (let i = 0; i < students.length; i++) {
  if (students[i].marks > 80) {
    highScorers[highScorers.length] = students[i];
  }
}
console.log(highScorers);
// Output:
// [
//   { name: 'Alice', age: 20, marks: 85 },
//   { name: 'Bob', age: 22, marks: 90 },
//   { name: 'David', age: 23, marks: 88 }
// ]
```

Explanation:

- The code checks if a student's marks are greater than 80.
- When the condition is true, the student object is added to the highScorers array.
- The first qualifying student is stored at index 0 (highScorers[0]), the second at index 1 (highScorers[1]), and so on.
- The logic ensures that only students with marks greater than 80 are appended to the new array in the correct order.

7. Aggregating Data

Manually calculate a value (e.g., total marks) by iterating through the array.

Example:
Calculate the total marks of all students.

```
let totalMarks = 0;
for (let i = 0; i < students.length; i++) {
  totalMarks += students[i].marks;
}
console.log(totalMarks);
// Output: 341
```

Explanation:

- totalMarks is initialized to 0.
- The loop iterates through the array, adding the marks of each student to totalMarks.
- This manually computes the sum of a specific property across all objects.

8. Deep Copying an Array of Objects

Create a new array where each object is a copy of the original objects.

Example:
Deep copy the students' array.

```
let copiedStudents = [];
for (let i = 0; i < students.length; i++) {
  let copiedStudent = { ...students[i] }; // Spread syntax to copy
object
  copiedStudents[i] = copiedStudent;
}
console.log(copiedStudents);
// Output: An identical copy of the `students` array.
```

Explanation:

- copiedStudents is a new array.

- Each object in students is copied using the spread operator { ...students[i]}, which ensures that changes to the copied array won't affect the original array.
- The copied objects are then stored in copiedStudents.

9. Finding Duplicates

Manually find duplicate objects based on a specific property.

Example:
Find duplicate student names.

```
let seenNames = {};
let duplicates = [];
for (let i = 0; i < students.length; i++) {
  let name = students[i].name;
  if (seenNames[name]) {
    duplicates[duplicates.length] = name;
  } else {
    seenNames[name] = true;
  }
}
console.log(duplicates);
// Output: Any duplicate names found.
```

Explanation:

- seenNames keeps track of names that have already been encountered.
- For each iteration, the name is checked in the seenNames object:
  1. If the name is already present, it's added to duplicates.
  2. Otherwise, the name is marked as seen.
- When a duplicate name is found, it is added to the duplicates array.
- For the first duplicate, duplicates.length is 0, so the name is assigned to duplicates[0].
- For the second duplicate, duplicates.length is 1, so the name is assigned to duplicates[1].
- This continues, ensuring each duplicate is stored in the next available index.
- This helps detect duplicate entries manually without built-in functions.

Best Practices

- Clear Logic: When not using helper functions, understand and implement the logic manually.
- Maintain Original Data: Use a separate array or objects when making modifications.
- Efficient Iteration: Optimize loops by exiting early (e.g., using a break for searches).
- Readable Code: Use descriptive variable names to clarify the purpose of each loop or action.
- This approach helps build a strong foundation for working with arrays of objects and prepares you to utilise JavaScript's built-in functions more effectively later.