Advanced JavaScript Patterns

Higher-Order Functions

Higher-order functions are a powerful paradigm in functional programming that allow you to write more abstract and reusable code. A higher-order function is one that takes other functions as arguments or returns a function as its result. This capability enables you to create flexible abstractions that can be adapted to many different use cases. The following example demonstrates a simple higher-order function that applies a given function to each element of an array.

```
function map(array, transform) {
  const result = [];
  for (let i = 0; i < array.length; i++) {
    result.push(transform(array[i]));
  }
  return result;
}
```

This implementation shows how a function can accept another function as a parameter. By passing different `transform` functions, you can implement various operations like doubling numbers, extracting properties from objects, or applying complex calculations. This abstraction makes the code more modular and testable.

Nested Control Flow

When dealing with complex data processing tasks, you often need to combine multiple control structures such as conditional statements and loops. The following example demonstrates a more sophisticated pattern that uses nested conditionals and loops to filter and process data with multiple criteria.

```
function processUserData(users, minAge, departments) {
  const results = [];
  for (let i = 0; i < users.length; i++) {
    const user = users[i];
    if (user.active === true) {
      if (user.age >= minAge) {
        let matched = false;
        for (let j = 0; j < departments.length; j++) {
          while (departments[j].length > 0) {
            const dept = departments[j].pop();
            if (user.department === dept) {
              matched = true;
              break;
```

```
            }
          }
          if (matched) break;
        }
        if (matched) {
          results.push(user);
        }
      }
    }
  }
  return results;
}
```

This function demonstrates proper whitespace preservation with indentation, nested conditional blocks, and multiple loop structures. When printing such code to PDF, maintaining consistent spacing and alignment is critical for readability and correctness verification.