Intermediate JavaScript Programming

LESSON 8: Code Quality & Tooling

Learning Objectives:

By the end of this lesson, participants will be able to:

- Use ESLint and Prettier to enforce consistent code style.
- Write clean, modular, and reusable functions.
- Organize code in ways that support maintainability and teamwork.

Lesson Outline:

I. The Value of Code Quality Tools (10 min)

Automated tools help catch issues early and enforce consistency across teams.

Benefits:

- Prevent small mistakes before they become bugs
- Enforce team conventions without argument
- Save time during code review

Example issue caught by ESLint:

```
if (isReady)
  start(); // Missing braces may be flagged
```

Example fix from Prettier:

```
const x=[1,2]; console.log(x);
// becomes:
const x = [1, 2];
console.log(x);
```

Commentary:

Code quality tools don't replace thinking — but they help reduce distractions and keep teams focused on substance.

II. Setting Up ESLint and Prettier (15 min)

1. **Install ESLint and Prettier locally**:

```
npm install eslint prettier --save-dev
```

2. **Initialize ESLint config**:

```
npx eslint --init
```

3. **Create `.prettierrc` for formatting rules**:

```
{
  "semi": true,
  "singleQuote": true
}
```

4. **Add scripts in `package.json`**:

```
"scripts": {
  "lint": "eslint .",
  "format": "prettier --write ."
}
```

Commentary:

Once configured, these tools can be run with one command and even integrated into your editor to apply fixes on save.

III. Writing Clean, Modular Functions (10 min)

Modular functions:

- Do one thing
- Have clear inputs and outputs
- Avoid modifying external state

Example:

```
function calculateArea(width, height) {
  return width * height;
}
```

Less clean version:

```
let lastResult;
function calculateArea(width, height) {
  lastResult = width * height; // modifies outer variable
  return lastResult;
}
```

Commentary:

Clean functions are easier to test and reuse. Side effects make behavior harder to trace.

IV. Organizing Larger Projects (10 min)

As projects grow, organizing by feature (rather than file type) can help.

Suggested folder structure:

```
src/
  components/
    Button.js
    Header.js
  utils/
    math.js
    format.js
  main.js
```

Avoid mixing unrelated concerns in the same file. Keep utility code in reusable modules.

Commentary:

Clear structure helps others (and future you) find what they need and make changes with confidence.

V. Editor and IDE Integration (10 min)

Most modern editors support ESLint and Prettier plugins:

- VS Code: install ESLint and Prettier extensions
- Enable "Format on Save"
- Use `.editorconfig` to set defaults for tabs, line endings, etc.

Example `.editorconfig`:

```
root = true

[*]
indent_style = space
```

```
indent_size = 2
end_of_line = lf
insert_final_newline = true
```

Commentary:

Consistent formatting helps teams avoid nitpicking and focus on solving problems.

VI. Recap & Q&A (5 min)

- Linting and formatting tools
- Modular function design
- Directory layout for clarity
- Editor setup for consistency

Final Multiple-Choice Question:

What is the purpose of using Prettier? A. To analyze runtime performance B. To detect security vulnerabilities C. To automatically format code consistently D. To compile JavaScript to machine code

(Answer: C. To automatically format code consistently)