Terminal Styling Practice Programs

How to Run These Programs:

1. Copy any program code below
2. Save it to a file (e.g., `practice.js`)
3. Run with Node.js: `node practice.js`
4. Experiment by modifying colors, text, and styles!

Note: Programs 6 and 11 have animations with delays and are commented out. Uncomment them to see the animations!

Table of Contents

Program 1: Basic Color Demo

What you'll learn: How to use basic ANSI color codes to color text in the terminal.

```
// ========================================
// PROGRAM 1: Basic Color Demo
// ========================================
console.log("\n=== PROGRAM 1: Basic Color Demo ===\n");

function program1_colorDemo() {
  // ANSI color codes
  const red = '\x1b[31m';
  const green = '\x1b[32m';
  const yellow = '\x1b[33m';
  const blue = '\x1b[34m';
  const magenta = '\x1b[35m';
  const cyan = '\x1b[36m';
  const reset = '\x1b[0m';

  console.log(`${red}This text is RED${reset}`);
  console.log(`${green}This text is GREEN${reset}`);
  console.log(`${yellow}This text is YELLOW${reset}`);
  console.log(`${blue}This text is BLUE${reset}`);
  console.log(`${magenta}This text is MAGENTA${reset}`);
  console.log(`${cyan}This text is CYAN${reset}`);
}

program1_colorDemo();
```

Program 2: Simple Logger

What you'll learn: Create a basic logger with different log levels (info, success, warning, error).

```
// ========================================
// PROGRAM 2: Simple Logger
// ========================================
console.log("\n=== PROGRAM 2: Simple Logger ===\n");

function program2_simpleLogger() {
  // Create a simple logger with different levels
  const logger = {
    info: (msg) => console.log(`\x1b[36m[INFO]\x1b[0m ${msg}`),
    success: (msg) => console.log(`\x1b[32m[SUCCESS]\x1b[0m ${msg}`),
    warning: (msg) => console.log(`\x1b[33m[WARNING]\x1b[0m ${msg}`),
    error: (msg) => console.log(`\x1b[31m[ERROR]\x1b[0m ${msg}`)
  };

  logger.info("Application starting...");
  logger.success("Connected to database");
  logger.warning("Low memory detected");
  logger.error("Failed to load configuration");
}

program2_simpleLogger();
```

Program 3: Text Styling Demo

What you'll learn: Apply text styles like bold, dim, italic, and underline.

```
// ========================================
// PROGRAM 3: Text Styling Demo
// ========================================
console.log("\n=== PROGRAM 3: Text Styling Demo ===\n");

function program3_textStyling() {
  const bold = '\x1b[1m';
  const dim = '\x1b[2m';
  const italic = '\x1b[3m';
  const underline = '\x1b[4m';
  const reset = '\x1b[0m';

  console.log(`${bold}This is BOLD text${reset}`);
  console.log(`${dim}This is DIM text${reset}`);
  console.log(`${italic}This is ITALIC text${reset}`);
  console.log(`${underline}This is UNDERLINED text${reset}`);

  // Combine styles
  const redBold = '\x1b[31m\x1b[1m';
  console.log(`${redBold}This is BOLD and RED${reset}`);
}

program3_textStyling();
```

Program 4: Background Colors

What you'll learn: How to add background colors to text and combine with foreground colors.

```
// ========================================
// PROGRAM 4: Background Colors
// ========================================
console.log("\n=== PROGRAM 4: Background Colors ===\n");

function program4_backgrounds() {
  const bgRed = '\x1b[41m';
  const bgGreen = '\x1b[42m';
  const bgYellow = '\x1b[43m';
  const bgBlue = '\x1b[44m';
  const white = '\x1b[37m';
  const black = '\x1b[30m';
  const reset = '\x1b[0m';

  console.log(`${bgRed}${white} Red Background ${reset}`);
  console.log(`${bgGreen}${black} Green Background ${reset}`);
  console.log(`${bgYellow}${black} Yellow Background ${reset}`);
  console.log(`${bgBlue}${white} Blue Background ${reset}`);
}

program4_backgrounds();
```

Program 5: Menu System

What you'll learn: Create a styled menu interface with borders and colors.

```
// ========================================
// PROGRAM 5: Menu System
// ========================================
console.log("\n=== PROGRAM 5: Menu System ===\n");

function program5_menuSystem() {
  const cyan = '\x1b[36m';
  const green = '\x1b[32m';
  const yellow = '\x1b[33m';
  const bold = '\x1b[1m';
  const reset = '\x1b[0m';

  console.log(`${cyan}${bold}╔════════════════════════════╗${reset}`);
  console.log(`${cyan}${bold}║       MAIN MENU            ║${reset}`);
  console.log(`${cyan}${bold}╚════════════════════════════╝${reset}`);
  console.log(`${green}1.${reset} Start Game`);
  console.log(`${green}2.${reset} Settings`);
  console.log(`${green}3.${reset} High Scores`);
  console.log(`${yellow}4.${reset} Exit`);
}

program5_menuSystem();
```

Program 6: Progress Bar

What you'll learn: Create an animated progress bar that updates in real-time.

Note: This program has a delay and is commented out by default. Uncomment to see the animation!

```
// ========================================
// PROGRAM 6: Progress Bar
// ========================================
console.log("\n=== PROGRAM 6: Progress Bar ===\n");

async function program6_progressBar() {
  const green = '\x1b[32m';
  const reset = '\x1b[0m';

  function showProgress(percent, label) {
    const filled = Math.floor(percent / 2);
    const empty = 50 - filled;
    const bar = '█'.repeat(filled) + '░'.repeat(empty);
    process.stdout.write(`\r${green}${bar}${reset} ${percent}% - ${label}`);
  }

  console.log("Downloading file...");

  for (let i = 0; i <= 100; i += 10) {
    showProgress(i, "Please wait...");
    await new Promise(resolve => setTimeout(resolve, 200));
  }

  console.log("\n✓ Download complete!");
}

// Uncomment to run (it has a delay):
// program6_progressBar();
```

Program 7: Status Messages

What you'll learn: Display status messages with colored symbols (✓, ✗, ⋯).

```
// ========================================
// PROGRAM 7: Status Messages
// ========================================
console.log("\n=== PROGRAM 7: Status Messages ===\n");

function program7_statusMessages() {
  const green = '\x1b[32m';
  const red = '\x1b[31m';
  const yellow = '\x1b[33m';
  const reset = '\x1b[0m';

  function status(message, type) {
    const symbols = {
      success: `${green}✓${reset}`,
      error: `${red}✗${reset}`,
      pending: `${yellow}⋯${reset}`
    };

    console.log(`${symbols[type]} ${message}`);
  }

  status("Server started successfully", "success");
  status("Database connection established", "success");
  status("Waiting for user input", "pending");
  status("Failed to load plugin", "error");
  status("Configuration file missing", "error");
}

program7_statusMessages();
```

Program 8: Timestamp Logger

What you'll learn: Add timestamps to log messages for better tracking.

```
// ========================================
// PROGRAM 8: Timestamp Logger
// ========================================
console.log("\n=== PROGRAM 8: Timestamp Logger ===\n");

function program8_timestampLogger() {
  const gray = '\x1b[90m';
  const cyan = '\x1b[36m';
  const reset = '\x1b[0m';

  function log(message) {
    const time = new Date().toLocaleTimeString();
    console.log(`${gray}[${time}]${reset} ${cyan}${message}${reset}`);
  }

  log("User logged in");
  log("Fetching data from API");
  log("Data processing complete");
}

program8_timestampLogger();
```

Program 9: Error Handler

What you'll learn: Create detailed error messages with type, message, and details.

```
// ========================================
// PROGRAM 9: Error Handler
// ========================================
console.log("\n=== PROGRAM 9: Error Handler ===\n");

function program9_errorHandler() {
  const red = '\x1b[31m';
  const yellow = '\x1b[33m';
  const bold = '\x1b[1m';
  const reset = '\x1b[0m';

  function handleError(errorType, message, details) {
    console.log(`\n${red}${bold}ERROR:${reset} ${errorType}`);
    console.log(`${yellow}Message:${reset} ${message}`);
    if (details) {
      console.log(`${yellow}Details:${reset} ${details}`);
    }
  }

  handleError("TypeError", "Cannot read property 'name' of undefined", "Line 42");
  handleError("ConnectionError", "Failed to connect to database", "Timeout after 5000ms");
}

program9_errorHandler();
```

Program 10: Table Display

What you'll learn: Create formatted tables with borders and colored data.

```
// ========================================
// PROGRAM 10: Table Display
// ========================================
console.log("\n=== PROGRAM 10: Table Display ===\n");

function program10_tableDisplay() {
  const cyan = '\x1b[36m';
  const green = '\x1b[32m';
  const yellow = '\x1b[33m';
  const bold = '\x1b[1m';
  const reset = '\x1b[0m';

  const users = [
    { id: 1, name: "Alice", status: "Active" },
    { id: 2, name: "Bob", status: "Active" },
    { id: 3, name: "Charlie", status: "Inactive" }
  ];

  // Header
  console.log(`${cyan}${bold}┌────┬─────────────┬──────────┐${reset}`);
  console.log(`${cyan}${bold}│ ID │    Name     │  Status  │${reset}`);
  console.log(`${cyan}${bold}├────┼─────────────┼──────────┤${reset}`);

  // Rows
  users.forEach(user => {
    const statusColor = user.status === "Active" ? green : yellow;
    console.log(`│  ${user.id} │ ${user.name.padEnd(11)} │ ${statusColor}${user.status.padEnd(8)}${reset} │`);
  });

  // Footer
  console.log(`${cyan}${bold}└────┴─────────────┴──────────┘${reset}`);
}

program10_tableDisplay();
```

Program 11: Loading Animation

What you'll learn: Create a spinning loading animation with different frames.

Note: This program has a 3-second animation and is commented out by default. Uncomment to see it!

```
// ========================================
// PROGRAM 11: Loading Animation
// ========================================
console.log("\n=== PROGRAM 11: Loading Animation ===\n");

async function program11_loadingAnimation() {
  const cyan = '\x1b[36m';
  const reset = '\x1b[0m';
  const frames = ['⠋', '⠙', '⠹', '⠸', '⠼', '⠴', '⠦', '⠧', '⠇', '⠏'];
  let i = 0;

  console.log("Loading animation demo:");

  const interval = setInterval(() => {
    process.stdout.write(`\r${cyan}${frames[i]}${reset} Loading...`);
    i = (i + 1) % frames.length;
  }, 100);

  setTimeout(() => {
    clearInterval(interval);
    process.stdout.write('\r✓ Complete!    \n');
  }, 3000);
}

// Uncomment to run (it has a 3-second animation):
// program11_loadingAnimation();
```

Program 12: Diff Viewer

What you'll learn: Display code differences like Git (removed lines in red, added lines in green).

```
// ========================================
// PROGRAM 12: Diff Viewer
// ========================================
console.log("\n=== PROGRAM 12: Diff Viewer ===\n");

function program12_diffViewer() {
  const green = '\x1b[32m';
  const red = '\x1b[31m';
  const reset = '\x1b[0m';

  function showDiff(oldLines, newLines) {
    console.log("Changes:");
    oldLines.forEach(line => {
      console.log(`${red}- ${line}${reset}`);
    });
    newLines.forEach(line => {
      console.log(`${green}+ ${line}${reset}`);
    });
  }

  const oldCode = ["const x = 10;", "console.log(x);"];
  const newCode = ["const x = 20;", "const y = 5;", "console.log(x + y);"];

  showDiff(oldCode, newCode);
}

program12_diffViewer();
```

Program 13: Colored JSON

What you'll learn: Display JSON data with syntax highlighting (keys, strings, numbers).

```
// ========================================
// PROGRAM 13: Colored JSON
// ========================================
console.log("\n=== PROGRAM 13: Colored JSON ===\n");

function program13_coloredJSON() {
  const blue = '\x1b[34m';
  const green = '\x1b[32m';
  const yellow = '\x1b[33m';
  const reset = '\x1b[0m';

  const data = {
    name: "John Doe",
    age: 30,
    isActive: true
  };

  console.log("{");
  console.log(`  ${blue}"name"${reset}: ${green}"${data.name}"${reset},`);
  console.log(`  ${blue}"age"${reset}: ${yellow}${data.age}${reset},`);
  console.log(`  ${blue}"isActive"${reset}: ${yellow}${data.isActive}${reset}`);
  console.log("}");
}

program13_coloredJSON();
```

Program 14: Box Messages

What you'll learn: Create boxed messages with borders for important announcements.

```
// ========================================
// PROGRAM 14: Box Messages
// ========================================
console.log("\n=== PROGRAM 14: Box Messages ===\n");

function program14_boxMessages() {
  const cyan = '\x1b[36m';
  const bold = '\x1b[1m';
  const reset = '\x1b[0m';

  function box(message, color = cyan) {
    const line = '─'.repeat(message.length + 2);
    console.log(`${color}${bold}┌${line}┐${reset}`);
    console.log(`${color}${bold}│${reset} ${message} ${color}${bold}│${reset}`);
    console.log(`${color}${bold}└${line}┘${reset}`);
  }

  box("Welcome to the Application!");
  console.log();
  box("Successfully saved!", '\x1b[32m');
}

program14_boxMessages();
```

Program 15: Complete Logger Class

What you'll learn: Build a full-featured logger class with timestamps and multiple log levels.

```
// ========================================
// PROGRAM 15: Complete Logger Class
// ========================================
console.log("\n=== PROGRAM 15: Complete Logger Class ===\n");

function program15_completeLogger() {
  class Logger {
    static colors = {
      reset: '\x1b[0m',
      gray: '\x1b[90m',
      red: '\x1b[31m',
      green: '\x1b[32m',
      yellow: '\x1b[33m',
      blue: '\x1b[34m',
      cyan: '\x1b[36m'
    };

    static timestamp() {
      return new Date().toISOString().split('T')[1].split('.')[0];
    }

    static log(level, color, message) {
      const time = `${this.colors.gray}[${this.timestamp()}]${this.colors.reset}`;
      const tag = `${color}[${level}]${this.colors.reset}`;
      console.log(`${time} ${tag} ${message}`);
    }

    static info(msg) {
      this.log('INFO', this.colors.cyan, msg);
    }

    static success(msg) {
      this.log('SUCCESS', this.colors.green, msg);
    }

    static warn(msg) {
      this.log('WARN', this.colors.yellow, msg);
    }

    static error(msg) {
      this.log('ERROR', this.colors.red, msg);
    }

    static debug(msg) {
      this.log('DEBUG', this.colors.blue, msg);
    }
  }

  // Usage example
  Logger.info("Starting application");
  Logger.debug("Loading configuration");
  Logger.success("Configuration loaded");
  Logger.warn("Using default settings");
  Logger.error("Could not connect to external service");
}

program15_completeLogger();

console.log("\n\n" + "=".repeat(50));
console.log("All programs completed!");
console.log("=".repeat(50) + "\n");
```

Practice Tips

- **Start with Program 1-4:** Learn the basics of colors and styles
- **Move to Program 5-10:** See practical applications
- **Try Program 11-15:** Build advanced features
- **Modify the code:** Change colors, add new features, combine different programs
- **Create your own:** Build a custom logger or CLI tool using what you learned!