COURSE TECHNOLOGY
CENGAGE Learning

JavaScript, Sixth Edition

Chapter 4
Debugging and Error Handling

Objectives

When you complete this chapter, you will be able to:

- Recognize error types
- Trace errors with dialog boxes and the console
- Use comments to locate bugs
- Trace errors with debugging tools
- Write code to respond to exceptions and errors

Introduction to Debugging

- Programming languages have syntax (rules)
- Logic
  - Order in which various program parts run (execute)
- Bug
  - Any program error
    - Causes program to function incorrectly due to incorrect syntax or flaws in logic
- Debugging
  - Process of tracing and resolving errors in a program

Understanding Syntax Errors

- Syntax errors
  - Occur when interpreter fails to recognize code
  - Causes
    - Incorrect use of JavaScript code
    - References to non-existent objects, methods, variables

Handling Run-Time Errors

- Run-time errors
  - Occur when interpreter encounters a problem while program executing
    - Not necessarily JavaScript language errors
  - Occur when interpreter encounters code it cannot execute
  - Run-time error can be caused by a syntax error

Identifying Logic Errors

- Logic errors
  - Flaw in a program's design
    - Prevents program from running as anticipated
  - "Logic" reference
    - Execution of program statements and procedures in the correct order to produce the desired results
  - Example: multiplying instead of dividing

```
var divisionResult = 10 * 2;
document.write("Ten divided by two is equal to "↵
  + divisionResult);
```

Identifying Logic Errors (cont'd.)

– Example: infinite loop

```
for (var count = 10; count >= 0; count) {
  document.write("We have liftoff in " + count);
}
```

– Example: infinite loop corrected

```
for (var count = 10; count >= 0; count--) {
  document.write("We have liftoff in " + count);
}
```


Interpreting Error Messages

- First line of defense in locating bugs
  - Browser console displays
    - Line number where error occurred
    - Error description
- Run-time errors
  - Error messages generated by a web browser
    - Can be caused by syntax errors but not by logic errors
  - Example:

```
function missingClosingBrace() {
  var message = "This function is missing a closing brace.";
  window.alert(message);
```


Interpreting Error Messages (cont'd.)

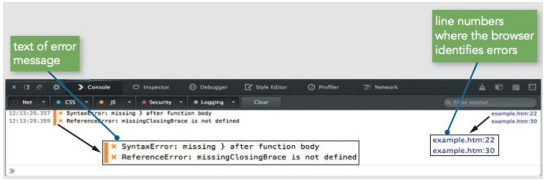


Figure 4-3 Firefox error messages

Interpreting Error Messages (cont'd.)

- Error message
  - Displays error's general location in a program
    - Not an exact indicator
- Browsers do not strictly enforce JavaScript syntax
- Mitigating bugs in JavaScript programs
  - Always use good syntax
  - Thoroughly test with every browser type, version
    - Test browser if used by more than one percent of the market

Using Basic Debugging Techniques

- Syntax errors can be difficult to pinpoint
- A few common techniques are helpful in tracking down bugs

Tracing Errors with the `window.alert()` Method

- Tracing
  - Examining statements in an executing program
- `window.alert()` method
  - A useful way to trace JavaScript code
  - Place at different points within program
    - Used to display variable or array contents or value returned from a function
- Use multiple `window.alert()` methods
  - Check values as code executes
- Example: function not returning correct result of 485
  - Returning 5169107

Tracing Errors with the `window.alert()` Method (cont'd.)

```
function calculatePay() {
  var payRate = 15; numHours = 40;
  var grossPay = payRate * numHours;
window.alert(grossPay);
  var federalTaxes = grossPay * .06794;
  var stateTaxes = grossPay * .0476;
  var socialSecurity = grossPay * .062;
  var medicare = grossPay * .0145;
  var netPay = grossPay - federalTaxes;
  netPay *= stateTaxes;
  netPay *= socialSecurity;
  netPay *= medicare;
  return netPay;
}
```


Tracing Errors with the `window.alert()` Method (cont'd.)

- Drawback
  - Must close each dialog box for code to continue executing
- Use selectively at key points
- Place debugging code at different indent level to distinguish from program code

Tracing Errors with the `console.log()` Method

- Trace a bug by analyzing a list of values
- Logging
  - writing values directly to the console using the `console.log()` method
  - syntax: console.log(*value*);
  - can log string literal, variable value, or combination
- Example
  - `calculatePay()` function

Tracing Errors with the `console.log()` Method (cont'd.)

```
function calculatePay() {
  var payRate = 15; numHours = 40;
  var grossPay = payRate * numHours;
console.log("grossPay is " + grossPay);
  var federalTaxes = grossPay * .06794;
  var stateTaxes = grossPay * .0476;
  var socialSecurity = grossPay * .062;
  var medicare = grossPay * .0145;
  var netPay = grossPay - federalTaxes;
console.log("grossPay minus federalTaxes is " + netPay);
  netPay *= stateTaxes;
console.log("netPay minus stateTaxes is " + netPay);
  netPay *= socialSecurity;
console.log("netPay minus socialSecurity is " + netPay);
  netPay *= medicare;
console.log("netPay minus medicare is " + netPay);
  return netPay;
}
calculatePay();
```


Tracing Errors with the `console.log()` Method (cont'd.)

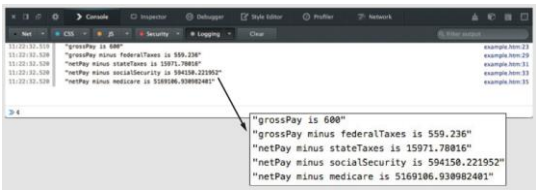


Figure 4-11 Contents of the console after executing the `calculatePay()` function

Tracing Errors with the `console.log()` Method (cont'd.)

- Driver program
  - simplified, temporary program
  - contains only the code you are testing

Using Comments to Locate Bugs

- Another method of locating bugs
  - "Comment out" problematic lines
- Helps isolate statement causing the error
- When error message first received
  - Start by commenting out only the statement specified by the line number in the error message
  - Continue commenting lines until error eliminated

Combining Debugging Techniques

- Combine debugging techniques
  - Aid in search for errors
- Example:
  - Use comments combined with an alert box or log message to trace errors in the `calculatePay()` function

Combining Debugging Techniques (cont'd.)

```
function calculatePay() {
   var payRate = 15;
   var numHours = 40;
   var grossPay = payRate * numHours;
window.alert(grossPay);
   // var federalTaxes = grossPay * .06794;
   // var stateTaxes = grossPay * .0476;
   // var socialSecurity = grossPay * .062;
   // var medicare = grossPay * .0145;
   // var netPay = grossPay – federalTaxes;
   // netPay *= stateTaxes;
   // netPay *= socialSecurity;
   // netPay *= medicare;
   // return Math.round(netPay);
}
```


Dependencies

- Relationship in which one statement depends on another statement executing successfully
- Can make debugging more challenging
- Important to retest program after fixing a bug to ensure other parts aren't affected by change

Tracing Errors with Debugging Tools

- Available in current versions of all modern browsers
  - Internet Explorer (IE)
  - Chrome
  - Firefox
- Accessible through same panel that opens when you use the console

Tracing Errors with Debugging Tools (cont'd.)

- Examining code manually
  - Usually first step taken with a logic error
  - Works fine with smaller programs
- Debugging tools
  - Help trace each line of code
  - More efficient method of finding and resolving logic errors

Understanding the IE, Firefox, and Chrome Debugger Windows

- Using Debugger Windows
  - Open a document to debug in a browser
  - Use keyboard shortcut or menu to open debugger

BROWSER	KEYBOARD SHORTCUT	MENU STEPS
Internet Explorer 9+	**F12**, then **Ctrl** + **3**	Click the **Tools** icon, click **F12 developer tools** on the menu, then in the window that opens, click the **Debugger** button
Firefox	**Ctrl** + **Shift** + **S** (Win) or **option** + **command** + **S** (Mac)	Click the **Open menu** button, click **Developer**, and then click **Debugger**
Chrome	**Ctrl** + **Shift** + **J** (Win) or **option** + **command** + **J** (Mac), then in the window that opens, click the **Sources** button	Click the **Customize and control Google Chrome** button, click **Tools**, click **JavaScript Console**, then in the window that opens, click the **Sources** button

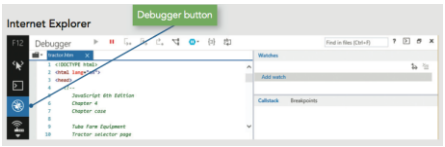
Table 4-1: Steps to open debuggers in IE, Firefox, and Chrome

Understanding the IE, Firefox, and Chrome Debugger Windows (cont'd.)

- Debugger window
  - Usually separate pane attached to bottom of browser window
  - Can also detach pane into separate window

Understanding the IE, Firefox, and Chrome Debugger Windows (cont'd.)

- Internet Explorer
  - Shows HTML code by default
  - Click View sources to select a different file



Understanding the IE, Firefox, and Chrome Debugger Windows (cont'd.)

- Firefox
  - Lists JavaScript files alphabetically
  - Click a filename to see its contents

Understanding the IE, Firefox, and Chrome Debugger Windows (cont'd.)

- Chrome
  - Displays no files by default
  - press Ctrl + O (Win) or command + O (Mac) to select from list of associated files

Setting Breakpoints

- Break mode
  - Temporary suspension of program execution
  - Used to monitor values and trace program execution
- Breakpoint
  - Statement where execution enters break mode
- When program paused at a breakpoint
  - Use debug tools to trace program execution

Setting Breakpoints (cont'd.)

- To set a breakpoint
  - Click the line number of the statement where execution should stop
- Resume button (Firefox/Chrome), Continue button (IE)
  - Executes rest of the program normally or until another breakpoint encountered

Setting Breakpoints (cont'd.)

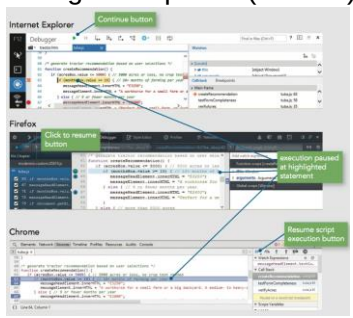


Figure 4-20 tuba.js execution stopped at the line 63 breakpoint

Clearing Breakpoints

- To clear a breakpoint
  - Click the line number
- To clear all breakpoints
  - Right-click any breakpoint
  - Click "Remove all breakpoints" or "Delete all"

Stepping Through Your Scripts

- Stepping into
  - Executes an individual line of code
    - Pauses until instructed to continue
  - Debugger stops at each line within every function
- Stepping over
  - Allows skipping of function calls
  - Program still executes function stepped over
- Stepping out
  - Executes all remaining code in the current function
  - Debugger stops at next statement in the calling function

Tracing Variables and Expressions

- Variables list
  - Displays all local variables within the currently executing function
  - Shows how different values in the currently executing function affect program execution
- Watch list
  - Monitors variables and expressions in break mode

Tracing Variables and Expressions (cont'd.)

- To access watch list
  - IE
    - Displayed by default on right side of pane
    - In break mode, local and global variables displayed
  - Firefox
    - Click Expand Panes button
    - Shows watch and variables list on right side of pane
  - Chrome
    - Displayed by default on right side of pane

Tracing Variables and Expressions (cont'd.)

- To add an expression to the watch list
  - Locate an instance of the expression in the program
  - Select it and copy it to the Clipboard
  - Click "Click to add" (IE) or "Add watch expression (Firefox or Chrome)
  - Paste expression from Clipboard
  - Press Enter

Examining the Call Stack

- Call stack
  - Ordered lists of which procedures (functions, methods, event handlers) have been called but haven't finished executing
- Each time a program calls a procedure:
  - Procedure added to top of the call stack

Examining the Call Stack

- IE and Chrome
  - Call stack list displayed to right of code
- Firefox
  - Call stack list displayed above code

Handling Exceptions and Errors

- Bulletproofing
  - Writing code to anticipate and handle potential problems
- One bulletproofing technique
  - Validate submitted form data
- Exception handling
  - Allows programs to handle errors as they occur in program execution
- Exception
  - Error occurring in a program

Throwing Exceptions

- Execute code containing an exception in a `try` statement
- `throw` statement
  - Specifies an error message in case an error that occurs within a `try` block

```
try {
   var lastName = document.getElementById("lName").value;
   if (lastName === "") {
      throw "Please enter your last name.";
   }
}
```


Catching Exceptions

- Use a `catch` statement
  - Handles, or "catches" the error
- Syntax:

```
catch(error) {
   statements;
}
```

- Example:

```
catch(lNameError) {
   window.alert(lNameError);
   return false;
}
```

Executing Final Exception Handling Tasks

- `finally` statement
  - Executes regardless of whether its associated `try` block throws an exception
  - Used to perform some type of cleanup
    - Or any necessary tasks after code evaluated with a `try` statement

Implementing Custom Error Handling

- Primary purpose of exception handling
  - Prevent users from seeing errors occurring in programs
  - Provide graceful way to handle errors
- Reason for using exception handling with JavaScript
  - Evaluate user input
- Programmers may write their own error-handling code
  - Can write user-friendly messages
  - Provides greater control over any errors

Implementing Custom Error Handling (cont'd.)

- Catching errors with the `error` event
  - Executes whenever error occurs on a Web page
  - Name of function to handle JavaScript errors
    - Assigned as event listener for `error` event
  - Preventing the Web browser from executing its own error handling functionality
    - Return return a value of true from the `error` event handler function

Implementing Custom Error Handling (cont'd.)

- Writing custom error-handling functions
  - JavaScript interpreter automatically passes three arguments to the custom error handling function
    - Error message, URL, line number
  - Use these values in custom error handling function
    - By adding parameters to the function definition
  - Use parameters in the function
    - Show a user the location of any JavaScript errors that may occur

Additional Debugging Techniques

- Includes
  - Checking HTML elements
  - Analyzing logic
  - Testing statements with console command line
  - Using the debugger statement
  - Executing code in strict mode
  - Linting
  - Reloading a Web page

Checking HTML Elements

- If a bug cannot be located using methods described in this chapter:
  - Perform a line-by-line analysis of the HTML code
  - Ensure all necessary opening and closing tags included
- Use code editor specialized for web development
  - Highlights syntax errors as you type
- Use the W3C Markup Validation Service to validate a Web page

Analyzing Logic

- Some JavaScript code errors stem from logic problems
  - Can be difficult to spot using tracing techniques
- Analyze each statement on a case-by-case basis

Testing Statements with the Console Command Line

- Console command line
  - Testing and executing JavaScript statements
    - Without HTML document or JavaScript source file
  - Useful if trying to construct the correct syntax for a mathematical expression
- Enter JavaScript statement at command line in web browser's console
- Including multiple statements at the command line
  - Separate statements with a semicolon

Using the `debugger` statement

- When you include the `debugger` statement in your code
  - web browser stops executing JavaScript code when it reaches the `debugger` statement
  - equivalent of a breakpoint that's part of your JavaScript code

Using Strict Mode

- Strict mode
  - Removes some features from JavaScript
  - Requires more stringent syntax for other features
    - Example: must always use `var` to declare variables
- Many removed or altered features in strict mode are known to cause hard to find bugs
- Include statement `"use strict";`
  - Including at start of script section requests strict mode for all code in that section
  - Including at start of code block in function requests strict mode just for that function

Linting

- Running code through a program that flags some common issues that may affect code quality
- jslint is a commonly used linting program
- Similar result to using strict mode, but generates a report containing line numbers

Reloading a Web Page

- Usually click the browser Reload or Refresh button
- Web browser cannot always completely clear its memory
  - Remnants of an old bug may remain
  - Force web page reload
    - Hold Shift key and click the browser's Reload or Refresh button
- May need to close browser window completely
- May need to delete frequently visited web pages

Summary

- Three types of program errors
  - Syntax errors, run-time errors, logic errors
- Error messages
  - First line of defense in locating bugs
- Tracing
  - Examination of individual statements in an executing program
- Using `console.log()` method to trace bugs
  - Helpful to use a driver program
- Browser debugging tools

Summary (cont'd.)

- Break mode
  - Temporary suspension of execution to monitor values and trace execution
- Breakpoint: statement in the code at which program execution enters break mode
- Stepping into, stepping over, and stepping out
- Variables list and watch list
- Call stack
  - List of procedures that have started but not finished

Summary (cont'd.)

- Bulletproofing
  - Writing code to anticipate, handle potential problems
- Exception handling
- `try`, `throw`, `catch`, `finally` statements
- JavaScript includes an `error` event
  - Executes whenever an error occurs on a web page
- Additional debugging methods and techniques
  - Checking HTML elements, analyzing logic, console command line, `debugger` statement, strict mode, linting, and reloading a web page