CHAPTER 4 – Debugging and Error Handling

Contents

[Introduction to Debugging 2](#_Toc13691252)

[Recognizing Syntax Errors: 2](#_Toc13691253)

[Recognizing Run-Time Errors 2](#_Toc13691254)

[Recognizing Logic Errors 2](#_Toc13691255)

[Interpreting Error Messages 2](#_Toc13691256)

[Using Basic Debugging Techniques 3](#_Toc13691257)

[Tracing Errors with the window.alert() Method 3](#_Toc13691258)

[Tracing Errors with the console.log() Method 3](#_Toc13691259)

[Using Comments to Locate Bugs 4](#_Toc13691260)

[Tracing Errors with Debugging Tools 4](#_Toc13691261)

[Setting Brake points: 5](#_Toc13691262)

[Stepping Through Your Scripts 5](#_Toc13691263)

[Tracing Variables and Expressions 6](#_Toc13691264)

[Examining the Call Stack 6](#_Toc13691265)

[Handling Exceptions and Errors 6](#_Toc13691266)

[Using the try and throw Statements: 6](#_Toc13691267)

[Catching Exceptions: 7](#_Toc13691268)

[Executing Final Exception Handling Tasks 7](#_Toc13691269)

[Implementing Custom Error Handling 7](#_Toc13691270)

[Additional Debugging Techniques 8](#_Toc13691271)

[Checking HTML elements 8](#_Toc13691272)

[Analyzing logic 9](#_Toc13691273)

[Testing statements with console command line 9](#_Toc13691274)

[Using the debugger statement 9](#_Toc13691275)

[Using Strict Mode 9](#_Toc13691276)

[Linting 9](#_Toc13691277)

[Reloading a Web page 9](#_Toc13691278)

## 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

### Recognizing Syntax Errors:

* Syntax errors
  + Occur when interpreter fails to recognize code
  + Causes

Incorrect use of JavaScript code

References to non-existent objects, methods, variables

### Recognizing 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

### Recognizing 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

### 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

## Using Basic Debugging Techniques

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

Example:

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;

window.alert(netPay);

netPay \*= stateTaxes;

window.alert(netPay);

return netPay;

}

### 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:

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);

return netPay;

}

“When using the console.log() method to trace bugs, it can be helpful to use a **driver program**, which is a simplified, temporary program that is used for testing functions and other code. A driver program is simply a JavaScript program that contains only the code you are testing. Driver programs do not have to be elaborate; they can be as simple as a single function 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; Once you eliminated the error message, you can examine the commented-out statements for the cause of the bug.

Combine debugging techniques - Aid in search for errors

“Any program longer than a handful of lines includes statements that depend on the successful execution of other statements or functions. These relationships, known as dependencies, can add an extra layer of complexity to debugging.”

## Tracing Errors with Debugging Tools

* Examining code manually

Usually first step taken with a logic error

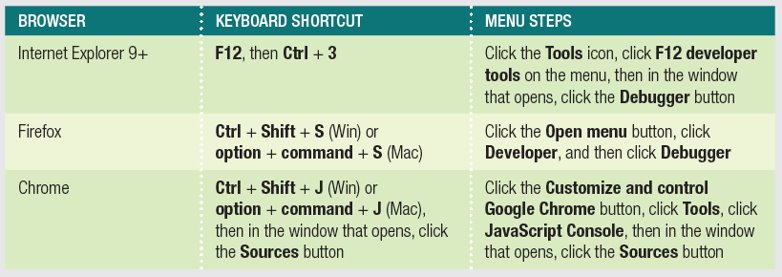
Works fine with smaller programs

* Debugging tools

Help trace each line of code

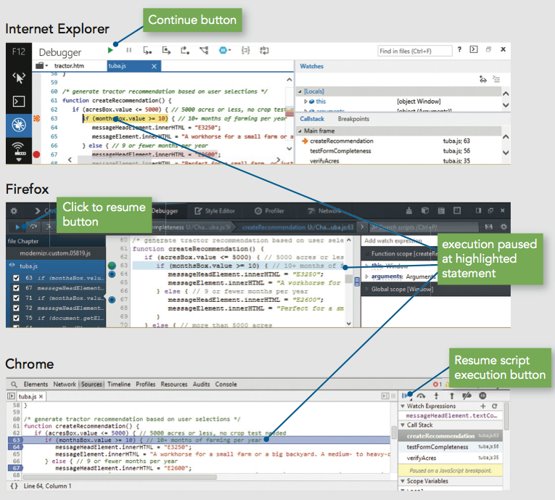
More efficient method of finding and resolving logic errors

Use keyboard shortcut or menu to open debugger:



### Setting Brake points:

* 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



### 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

### 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 and then removed after it finishes executing.

## Handling Exceptions and Errors

* Bulletproofing - Writing code to anticipate and handle potential problems
  + One bulletproofing technique is to validate submitted form data. For example, you might run a script when the Submit button is clicked that verifies that the value of any required text box isn’t an empty string.
  + Exception handling another bulletproofing method that allows programs to handle errors as they occur in program execution

“JavaScript includes four statements specifically for exception handling: try, throw, catch, and finally.”

### Using the try and throw Statements:

You enclose code that may contain an exception in a **try statement**.

If the result of evaluating statements is an error, then an error is triggered, or **thrown**.

Example:

try {

var lastName = document.getElementById("lName").value;

if (lastName === "") {

throw "Please enter your last name.";

}

}

### Catching Exceptions:

The **catch** statement accepts a single argument that assigns a name to the error message specified by the exception thrown by a **try** statement. Whatever argument name you specify when creating the **catch** statement is the variable name you use within statements to refer to the text of the thrown exception.

Example:

catch (lNameError) {

window.alert(lNameError);

return false;

}

### Executing Final Exception Handling Tasks

JavaScript’s exception handling functionality also includes a **finally statement** that executes regardless of whether it’s associated try block throws an exception. You normally use a finally statement to perform some type of cleanup or any necessary tasks after code is evaluated with a try statement.

Example:

finally {

lNameValid = true;

}

### 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

Catching Errors with the error Event

JavaScript includes an error event that executes whenever an error occurs on a web page. The following code specifies a function named processErrors() to handle any JavaScript errors that occur on a page.

if (window.addEventListener) {

window.addEventListener("load", processErrors, false);

} else if (window.attachEvent) {

window.attachEvent("onerror", processErrors);

}

To prevent a web browser from executing its own error-handling functionality, you return a value of true from the error event function, as demonstrated in the following example of the processErrors() function:

*function processErrors() {*

*//custom error handling code...*

*return true;*

*}*

Writing Custom Error-Handling Functions

When you specify a custom error-handling function by assigning it to the error event, the JavaScript interpreter automatically passes three arguments in the following order to the function for any JavaScript errors that occur: error message, URL, and line number. You can use the values in your custom error-handling function by adding parameters to the function definition. You can then use the parameters in your function to send information to the console about the location of any JavaScript error that may occur.

For example the following code shows the processErrors() function containig parameters that are assigned the three arguments that are passed by the JavaScript interpreter.

function processErrors(errorMessage, errURL, errLineNumber) {

console.log("The file " + errURL + " generated the following error: " + errorMessage + " on line " +

errLineNumber);

return true;

}

if (window.addEventListener) {

window.addEventListener("error", processErrors, false);

} else if (window.attachEvent) {

window.attachEvent("onerror", processErrors);

}

## Additional Debugging Techniques

### 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.

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 console command line

If you find that the error in your code is the result of a single statement, you can test the statement using the console command line without rerunning the entire program. This is useful if trying to conduct the correct syntax for a mathematical expression.

### 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

Starting with version 5.1 ECMAScript included a new feature: the ability to request processors to treat code using strict mode. In strict mode, some features are removed from the language, while other features require more stringent syntax. For instance, declaring a variable without the var keyword triggers an exception. While coding in strict mode may in fact generate more errors, it can be a helpful debugging tool.

### 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.