How to test and debug a JavaScript application
Objectives

Applied

• Use Firefox to find the cause of any syntax, runtime, or logic errors.

• Display error messages in the Firefox browser and in any other browsers that you’re testing your application with.

• Trace the execution of an application with alert statements.

• Use the Firefox to view error messages, view the stack trace, profile the execution of an application, and test short segments of code in the command line.

• Use FireFox to set breakpoints, to step through the execution of the statements in an application, to view the current variables, and to set and view watch expressions.
Objectives (continued)

Knowledge

- Describe the three testing phases for a JavaScript application.
- Distinguish between syntax, runtime, and logic errors.
- Describe the use of stack traces, breakpoints, and watch expressions.
- Distinguish between the three types of Step commands that you can use to step through a program: Step Into, Step Over, and Step Out.
Sample Application

Area and Perimeter Calculator

Enter the values below and click "Calculate".

Length: 
Width: 
Area: 
Perimeter: 

[Calculate]
The goal of testing

- To find all errors before the application is put into production.

The goal of debugging

- To fix all errors before the application is put into production.
Three test phases

- Check the user interface to make sure that it works correctly.
- Test the application with valid input data.
- Test the application with invalid data or unexpected user actions.

The three types of errors that can occur

- Syntax errors
- Runtime errors
- Logic errors
JavaScript code that contains errors

```javascript
var $ = function (id) {
    return document.getElementById(id);
}

var calculate = function () {
    var length = parseFloat( $('#length').value );
    var width = parseFloat( $('#width').value );
}
```
Common syntax errors

- Misspelling keywords.
- Forgetting an opening or closing parenthesis, bracket, brace, or comment character.
- Breaking a single line into two valid statements where JavaScript inserts a semicolon.
- Forgetting to use a semicolon that is essential to the logic of the code.
- Forgetting an opening or closing quote mark.
- Not using the same opening and closing quote mark.
Problems with identifiers

- Misspelling or incorrectly capitalizing an identifier.
- Using a reserved word, global property, or global method as an identifier.
Problems with values

- Not checking that a value is the right data type before processing it.
- Forgetting to use the parseInt or parseFloat function to convert a user entry into a numeric value.
- Using one equal sign instead of two when testing for equality.

A problem with floating-point arithmetic

- The number data type in JavaScript uses floating-point numbers and that can lead to arithmetic math errors.
- For example, 0.2 + 0.7 in JavaScript is 0.8999999999999999.
- One way around this is to round the number to the desired decimal place with the toFixed method and then convert it back to a floating-point number with the parseFloat method.
A problem with comparing values of different data types

- When you compare two values that are of different data types, JavaScript will internally convert both values to numbers.
- In this conversion, an empty string is converted to 0; true is converted to 1; and false is converted to 0.
- This can lead to unexpected result when using the equality operator. For example, the expression 0 == "" will evaluate to true since an empty string is converted to 0.
How to display the Web Console

- Use the Tools → Web Developer → Web Console

How to display the source code

- Click on the link on the right or click on Debugger.
The source code that’s displayed when you click on the link or Debugger.
JavaScript with alert statements that trace the execution of the code

```javascript
var calculate_click = function () {
    var length = parseFloat( $('#length').value );
    alert("User entered the length = " + length);
}
```
Use the alert dialog box to check values
Set a breakpoint

How to set a breakpoint

- Click in the bar to the left of a statement.
- To set a conditional breakpoint, right-click on a statement and enter a conditional expression in the resulting dialog box.
How to step through JavaScript code from a breakpoint

- Click the Step Into button to step through the code one line at a time.
- Click the Step Over button to execute any called functions without stepping through them.
- Click the Step Out button to execute the rest of a function without stepping through it.
- Click the Continue button to resume normal execution.
How to see the value of variables at each step

- At each step, the Watch tab displays the values of the current variables.
- You can also hover the mouse cursor over a variable name to display its current value.

How to set a watch expression

- Click “New watch expression…” in the Watch tab and type the variable name or the expression that you want to watch.

How to disable or remove breakpoints

- To remove a breakpoint, click on the red breakpoint marker in the Script tab, or click on the close button for the breakpoint in the Breakpoints tab.
- To disable a breakpoint, click on the check box for it in the Breakpoints tab.
The Safari console with an error message

How to display the Safari console

- Display the Develop menu.
- To do that, use the Edit ➔ Preferences command, click the Advanced tab, and check the “Show Develop menu in menu bar” checkbox.
- Then, to display the console, use the Develop ➔ Show Error Console command.
The Safari console when you click on the URL in a message

```javascript
return document.getElementById(id);

var calculate_click = function () {
    var investment = parseFloat($("investment").value);
    var annualRate = parseFloat($("rate").value);
    var years = parseInt($("years").value);
    $("futureValue").value = "";

    var monthlyRate = annualRate / 12 / 100;
    var months = years * 12;
    var futureValue = 0;
    for (i = 1; i <= months; i++) {
        futureValue = (futureValue + investment) * (1 + monthlyRate);
    }
    futureValue = Math.round(futureValue * 100) / 100;
    if (!isNaN(futureValue)) {
        var fvDisplay = "$" + futureValue;
        var cents = Math.round((futureValue - parseInt(futureValue)) * 100);
        if (cents == 0) {
            fvDisplay += ",00";
        } else if (cents % 10 == 0) {
```
The Opera error console with an error message

```
Event thread: load
Error:
  name: TypeError
  messages: Statement on line 3: Type mismatch |usually non-object value supplied where object required|
Backtrace:
  Line 2 of linked script http://192.168.1.42/javascript/chapter_03/futureValue_error.js
  return document.getElementById(id);
  Line 32 of linked script http://192.168.1.42/javascript/chapter_03/futureValue_error.js
  $("calculate").on('click' = calculate_click;
...
stacktrace: n/a: see 'opera:config#UserPrefs|Exceptions Have Stacktrace'
```
The Opera developer tools

How to open the error console and the developer tools

- To open the error console, use the Tools→Advanced→Error Console command.
- To open the developer tools, use the Tools→Advanced→Developer Tools command.
The Chrome browser with the Developer menu

How to open the JavaScript console
- Select the Developer → JavaScript Console command.

How to open the debugger
- Select the Developer → Debug JavaScript command.
The JavaScript console with an error message

```javascript
var calculate_click = function () {
    var investment = parseFloat ($("investment").value);

    var annualRate = parseFloat ($("rat").value);
    var years = parseInt ($("years").value);

    $("futureValue").value = "";

    if (isNaN(investment) || investment <= 0) {
        alert("Investment must be a valid number\nand greater than zero." );
    } else if (isNaN(annualRate) || annualRate <= 0) {
        alert("Annual rate must be a valid number\nand greater than zero." );
    }
};
```