Chapter 5

How to test and debug
a JavaScript application

Objectives

- An introduction to testing and debugging
- How to debug with Chrome’s developer tools
- Other debugging methods

An introduction to testing and debugging

An introduction to testing and debugging

- **Testing** an application that you run it and make sure that it works correctly.
- **Debugging** an application, that you fix the errors(bugs) that you discover during testing.

An introduction to testing and debugging (cont.)

- **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.

An introduction to testing and debugging (cont.)

- The three types of errors that can occur
  - Syntax errors
  - Runtime errors
  - Logic errors

Common JavaScript errors

Common syntax errors

- Misspelling keywords, like coding getElementByID instead of getElementById.
- Omitting required parentheses, quotation marks, or braces.
- Not using the same opening and closing quotation mark.
- Omitting the semicolon at the end of a statement.
- Misspelling or incorrectly capitalizing an identifier, like defining a variable named salesTax and referring to it later as salestax.

Problems with HTML references

- Referring to an attribute value or other HTML component incorrectly, like referring to an id as salesTax when the id is sales_tax.

Problems with data and comparisons

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

Common JavaScript errors(cont.)

Problems with floating-point arithmetic

- The number data type in JavaScript uses floating-point numbers, and that can lead to arithmetic results that are imprecise. For example,

```
var salesAmount = 74.95;
salesTax = salesAmount * .1;                    // result is 7.495000000000001
```

- One way to fix this potential problem is to round the result to the right number of decimal places and then convert it back to a floating-point number:

```
salesTax = salesTax.toFixed(2)                  // result is 7.50 as a string
salesTax = parseFloat(salesTax.toFixed(2));     // result is 7.50 as a number
```

Problems with undeclared variables that are treated as global variables

- If you assign a value to a variable that hasn't been declared, the JavaScript engine treats it as a global variable. This can happen when you misspell a variable name, as in this example:

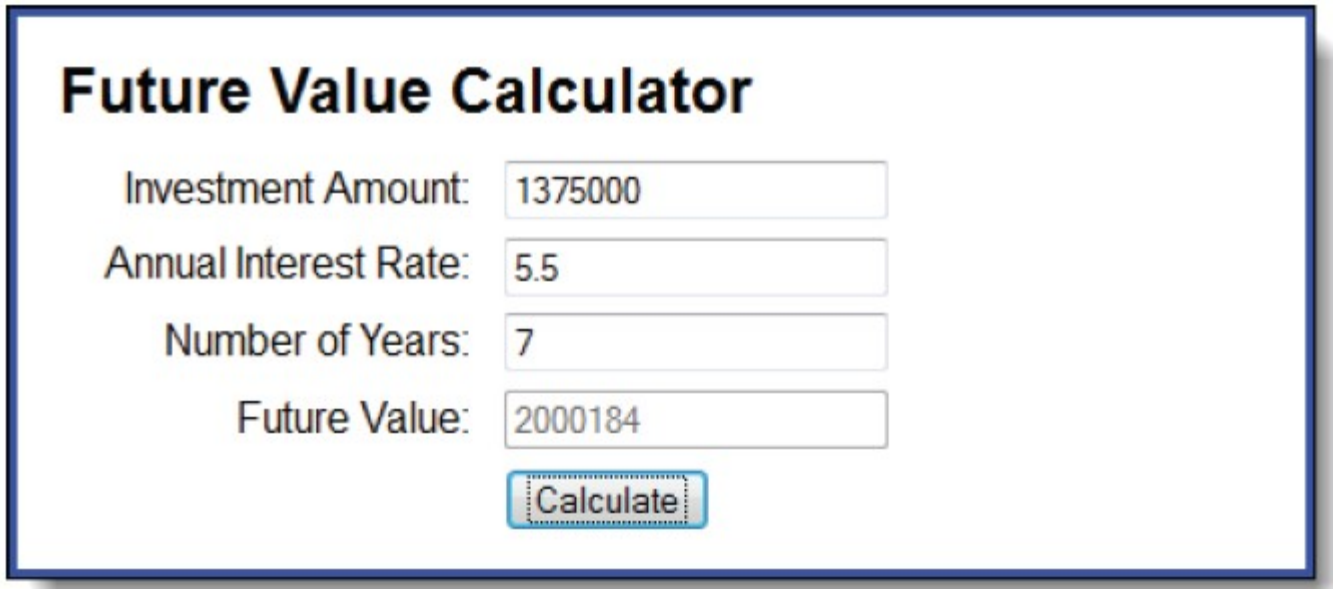
```
var calculateTax = function (subtotal, taxRate) {
    var salesTax = subtotal * taxRate;              // salesTax is local
    salestax = parseFloat(salesTax.toFixed(2));     // salestax is global
    return salesTax;          // salesTax isn't rounded but salestax is
}
```

How to plan the test run

- There are at least two test phase:
  - 1st phase: test application with valid data
  - 2nd phase: test application with invalid data

How to plan the test run (cont.)

- Try to test with Future Value application



How to plan the test run (cont.)

- Two common testing problems
  - Not testing a wide enough range of entries
  - Not knowing what the results of each set of entries should be and assuming that the results are correct because they look correct.

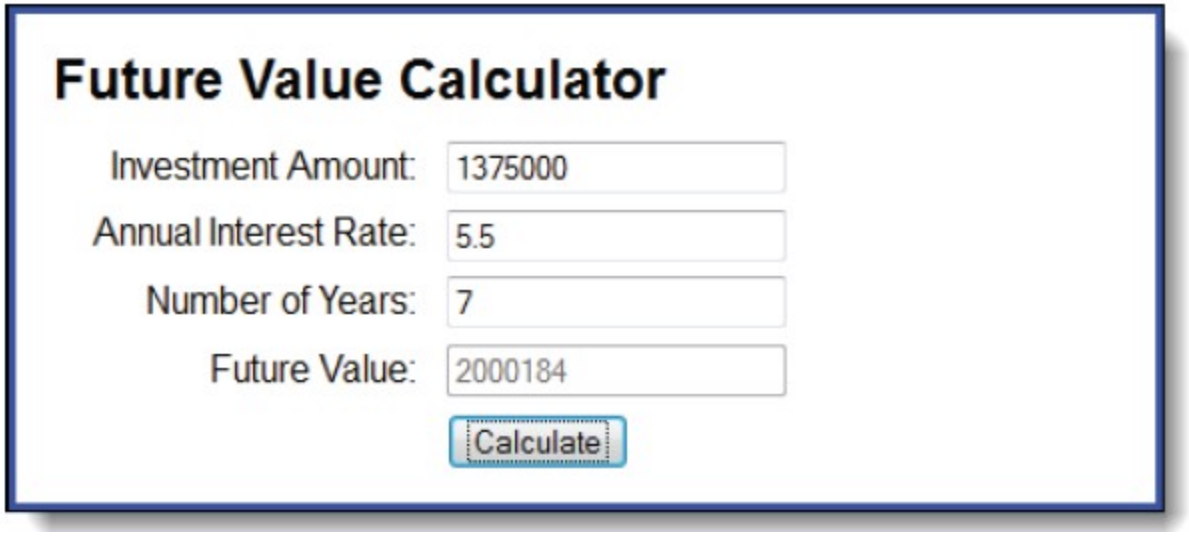
How to use top-down coding and testing simplify debugging

- **Top-down coding and testing** that you start by coding and testing a small portion of code.
- You can build first with small code then add more code to test for next phase.

How to use top-down coding and testing simplify debugging (cont.)

- Look at example

The user interface for a Future Value application



How to use top-down coding and testing simplify debugging (cont.)

Testing phase 1: No data validation

```
var $ = function (id) {
    return document.getElementById(id);
}
var calculateClick = function () {
    var investment = parseFloat( $("investment").value );
    var annualRate = parseFloat( $("rate").value );
    var years = parseInt( $("years").value );
    for ( i = 1; i <= years; i++ ) {
        investment += investment * annualRate / 100;
    }
    $("future_value").value = investment.toFixed();
}
window.onload = function () {
    $("calculate").onclick = calculateClick;
}
```

How to use top-down coding and testing simplify debugging (cont.)

Testing phase 2: Add data validation for just the first entry

```
if (isNaN(investment) || investment <= 0) {
    alert("Investment must be a number and greater than zero.");
}
else {
    // the future value calculation from phase 1
}
```

Testing phase 3: Add data validation for the other entries

```
// Add data validation for the other entries
```

Testing phase 4: Add the finishing touches

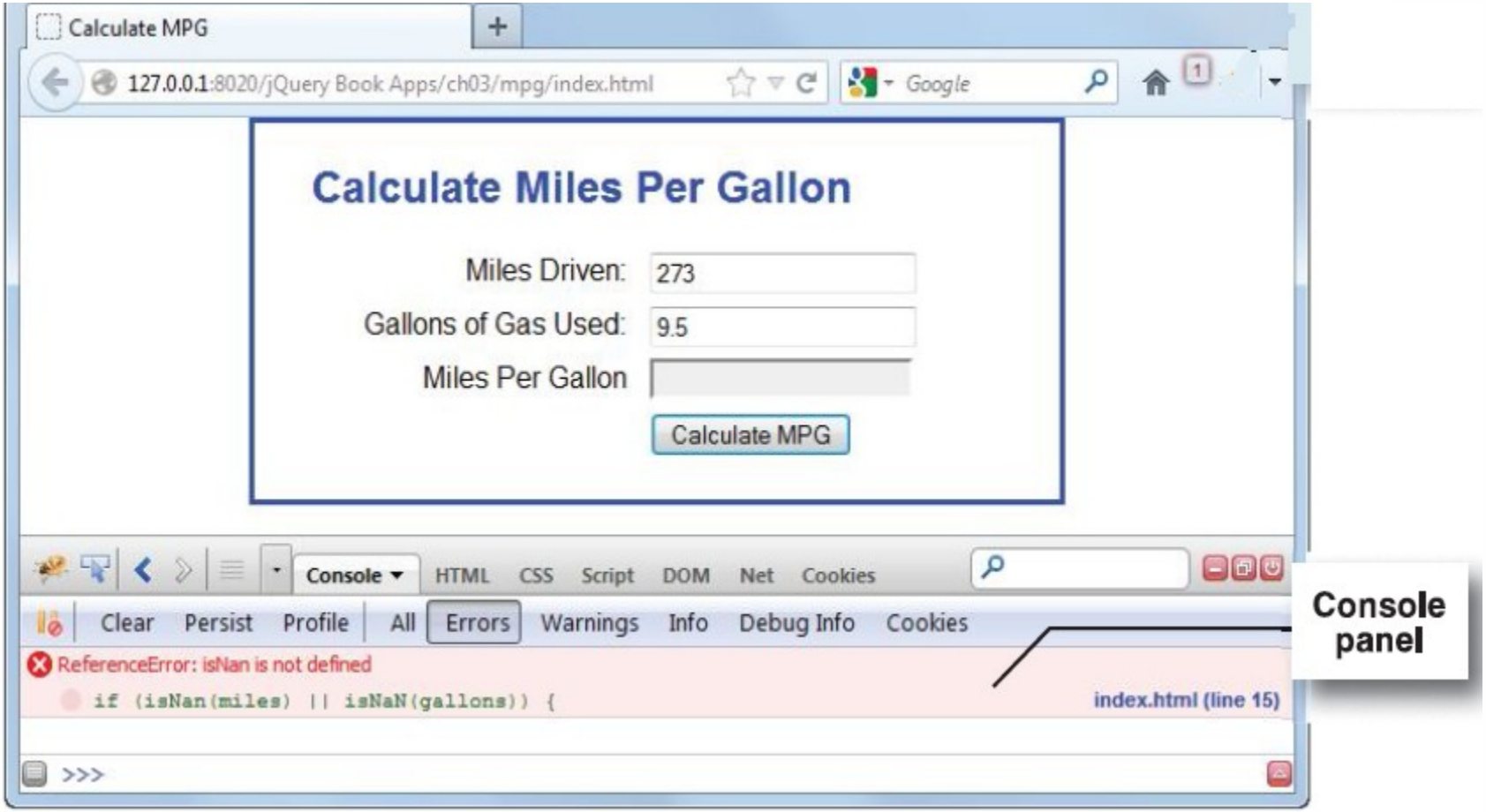
```
// Add finishing touches like moving the focus to the first text box
```

How to debug with Chrome’s developer tools

How to debug with Chrome’s developer tools

- Chome’s developer tools provide some excellent debugging features, like identifying the JavaScript statement that caused an error.
- Press F12 key to start the chrome’s development tools.

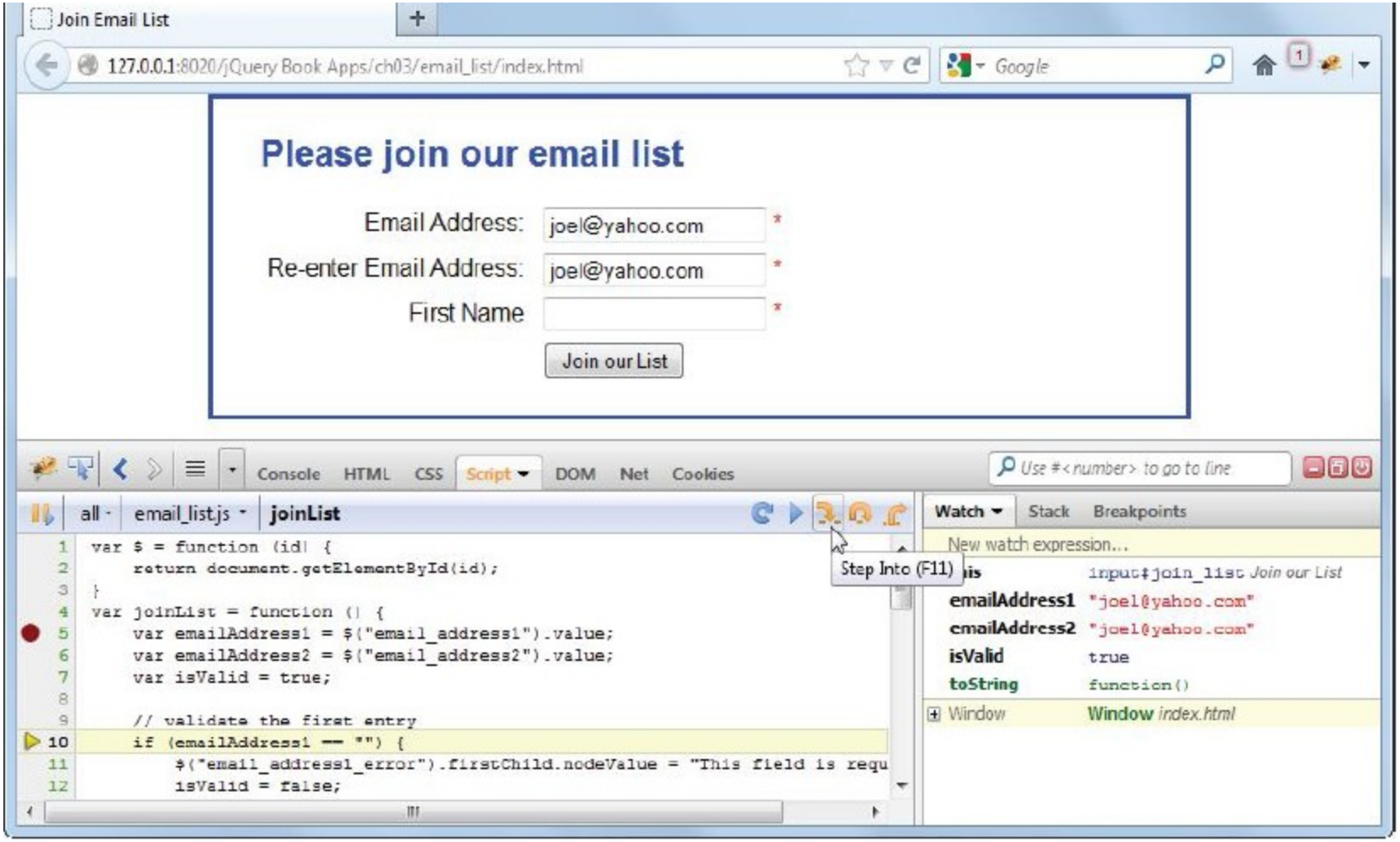
How to debug with Chrome's developer tools (cont.)



How to use breakpoints and step through your code

- You can set a breakpoint to stop the execution of your application.
- The execution of application will stop before statement that is set breakpoint.
- You can see value of variables in watch windows. They can help you define caused of error.

How to use breakpoints and step through your code (cont.)



How to use breakpoints and step through your code (cont.)

- You can step through the JavaScript code by press the keys below.

Button	Key	Description
Step Into	F11	Step through the code one line at a time.
Step Over	F10	Run any called functions without stepping through them.
Step Out	SHIFT + F11	Execute the rest of a function without stepping through it.
Resume	F8	Resume normal execution.

Other debugging methods

How to trace the execution of your JavaScript code

- You can insert console().log method to display values of key variable at a key point to console panel.
- Or you can use aler() method to display values of key variable at a key point to popups.
- Base on the value of key variable you can determine the caused of error.

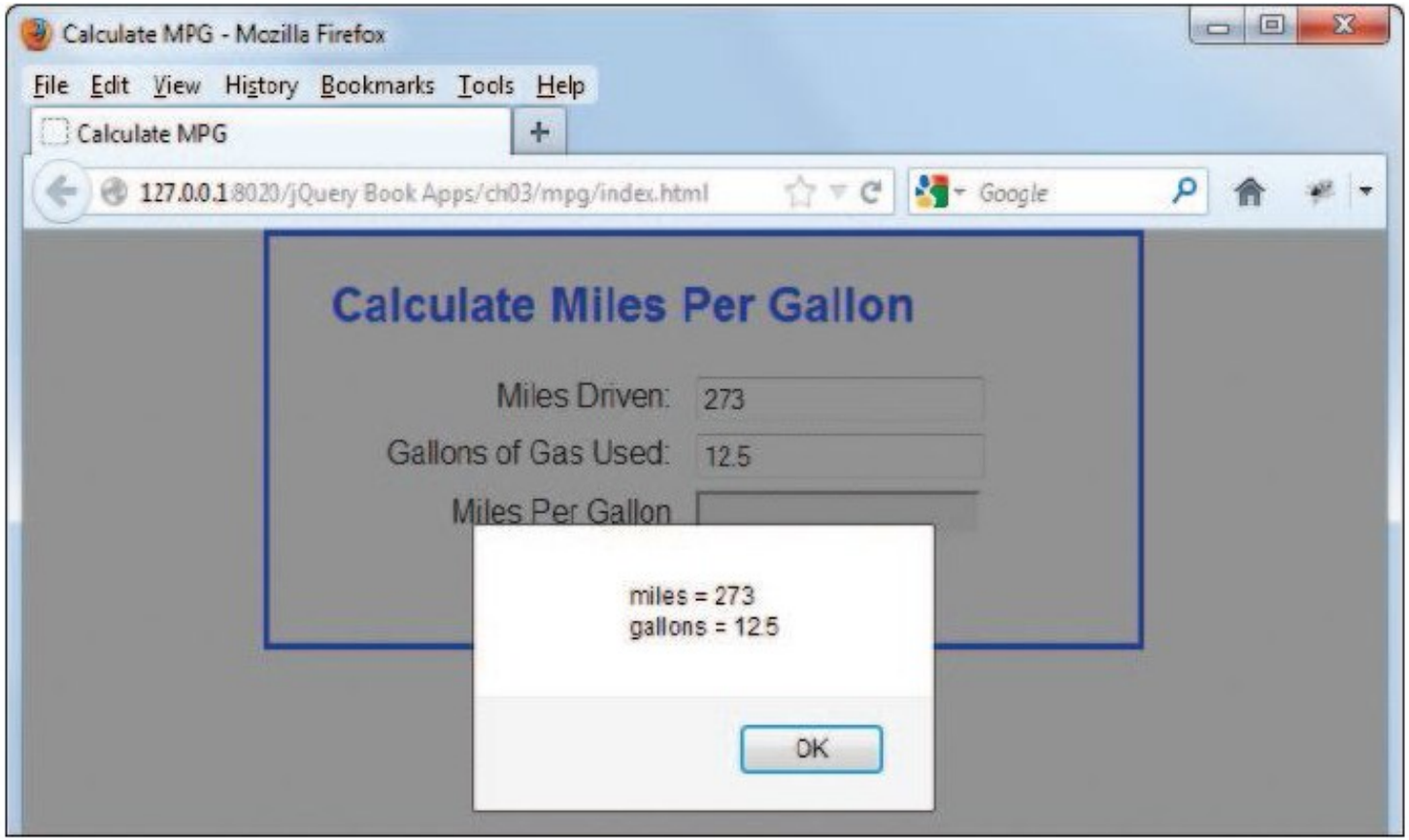
How to trace the execution of your JavaScript code (example)

JavaScript with five alert statements that trace the execution of the code

```
var $ = function (id) {
    alert("$ function has started");
    return document.getElementById(id);
}
calculateMpg = function () {
    alert("calculateMpg function has started");
    var miles = parseInt($("miles").value);
    var gallons = parseFloat($("gallons").value);
    alert("miles = " + miles +
          "\ngallons = " +  gallons);

    if (isNaN(miles) || isNaN(gallons)) {
        alert("Both entries must be numeric");
    }
    else {
        alert("The data is valid and the calculation is next");
        var mpg = miles / gallons;
        $("mpg").value = mpg.toFixed(1);
    }
}
window.onload = function () {
    alert("onload function has started");
    $("calculate").onclick = calculateMpg;
    $("miles").focus();
}
```

How to trace the execution of your JavaScript code (example)



How to view the source code

- You can view the source code of HTML and JavaScript to identify an error.
- Right click on page and select View Source or View Page Source to display the source code of page.

How to validate the HTML

- W3C provide a tool permit you validate your html file at http://validator.w3.org/
- This tool will check and report the error in HTML file to you. Then you can fix it.

Summary

- **Testing** an application that you run it and make sure that it works correctly.
- **Debugging** an application, that you fix the errors(bugs) that you discover during testing.
- The three types of errors that can occur: Syntax errors, Runtime errors,Logic errors.
- There are at least two test phase:
  - 1st phase: test application with valid data
  - 2nd phase: test application with invalid data
- Chome’s developer tools provide some excellent debugging features, like identifying the JavaScript statement that caused an error.

Summary(cont.)

- You can set a breakpoint to stop the execution of your application.
- The execution of application will stop before statement that is set breakpoint.
- You can see value of variables in watch windows. They can help you define caused of error.
- You can step through the JavaScript code to identify the caused of error.
- You can use trace of code, view source or HTML validator tool of W3C to debugging an application.