Chapter 14

How to work with control structures, Exceptions, and regular expressions

Objectives

- More about control structures
- The Invoice application
- How to handle exceptions
- How to use regular expressions
- The Account Profile application

What else you need to know about control structures

How to use the equality and identity operators

- The equality operators

Operator	Description	Example
`==`	Equal	`lastName == “Hopper”`
`!=`	Not equal	`Months !=0`

- The identity operators

Operator	Description	Example
`===`	Equal	`lastName === “Hopper”`
`!==`	Not equal	`Months !==0`

How to use the equality and identity operators (cont.)

- The ***equality operators*** perform type ***coercion*** whenever that's necessary. That means they convert data from one type to another before perform operator.
- The ***identity operators*** do not perform type coercion. If the two operands are different types, the result will always be false.

How to use the break and continue statements

- The ***break statement*** ends a loop.
- The ***continue statement*** ends the current iteration of a loop, but allow the next iteration to proceed.

How to use the break and continue statements (cont.)

- **Example 1**: The break statement in a while loop

```
var number;
while(true){
   number = parseInt(prompt("Enter a number
                 from 1 to 10."));
   if(isNaN(number)||number<1||number>10){
      alert("Invalid entry. Try again");
   }else {
      break;
   }
}
Alert(number);
```

How to use the break and continue statements (cont.)

- **Example 2**: The continue statement in a while loop

```
var sum=0;
var number=0;
while(number<=40){
    number++;
    if(number %5 !==0){
        continue;}
    sum +=number;
}
```

How to use the switch statements

- The syntax of switch case

```
switch(expression){
    case value1:
        statements;
        break;
    case value2:
        statements;
        break;
    …
    case valuen:
        statements;
        break;
    default:
        statements;
}
```

How to use the switch statements (cont.)

- A switch statement with a default case

```
switch( letterGrade ){
    case “A”:
        message = “well above average”;
        break;
    case “B”:
        message = “above average”;
        break;
    case “C”:
        message = “average”;
        break;
    case “D”:
        message = “below average”;
        break;
    default:
        message = “invalid grade”;
}
```

How to use the switch statements (cont.)

- A switch statement with fall through

```
switch( letterGrade ){
    case “A”:
    case “B”:
        message = “Scholarship approved”;
        break;
    case “C”:
        message = “Application requires review”;
        break;
    case “D”:
    case “F”:
        message = “Scholarship not approved”;
        break;
}
```

How to use the condition operator

- Syntax of the condition operator

```
(conditional_expression) ? Value_if_true:value_if_false
```

- Example 1: Setting a string based on a comparison

```
var message =(age>=18)? ”can vote” : “can’t vote”;
```

- Example 2: calculating overtime pay

```
var overtime =(hours>=40)? (hours-40)*rate*1.5:0;
```

- Example 3: Returning one of two values based on comparison

```
return (number >higest)? higest:number;
```

How to use the AND and OR operators for selections

- Example 1: An OR selection that returns a Boolean value

```
var selected =(state===“CA” || state===“NC”);
```

- Example 2: An AND selection that returns a null, undefined, or String value

```
var selected = state && state.toString();
```

- Example 3: An OR selection that won’t return null or undefined value

```
var selected = state ||”CA”;
```

- Example 4: An OR selection that set default value

```
var selected = dt || new Date();
```

How to use the AND and OR operators for selections (cont.)

- These

```
var selected =(state===“CA” || state===“NC”);
```

- Example 2: An AND selection that returns a null, undefined, or String value

```
var selected = state && state.toString();
```

- Example 3: An OR selection that won’t return null or undefined value

```
var selected = state ||”CA”;
```

- Example 4: An OR selection that set default value

```
var selected = dt || new Date();
```

How to use the AND and OR operators for selections (cont.)

- How to rewrite selections
  - Example 1:

```
var selected =false;
if(state===“CA” || state===“NC”){
    selected =true;
}
```

  - Example 2:

```
var selected;
if(state){
    selected =sate.toString();
}
```

How to use the AND and OR operators for selections (cont.)

- How to rewrite selections
  - Example 3:

```
var selected =“CA”;
if(state){
    selected =state;
}
```

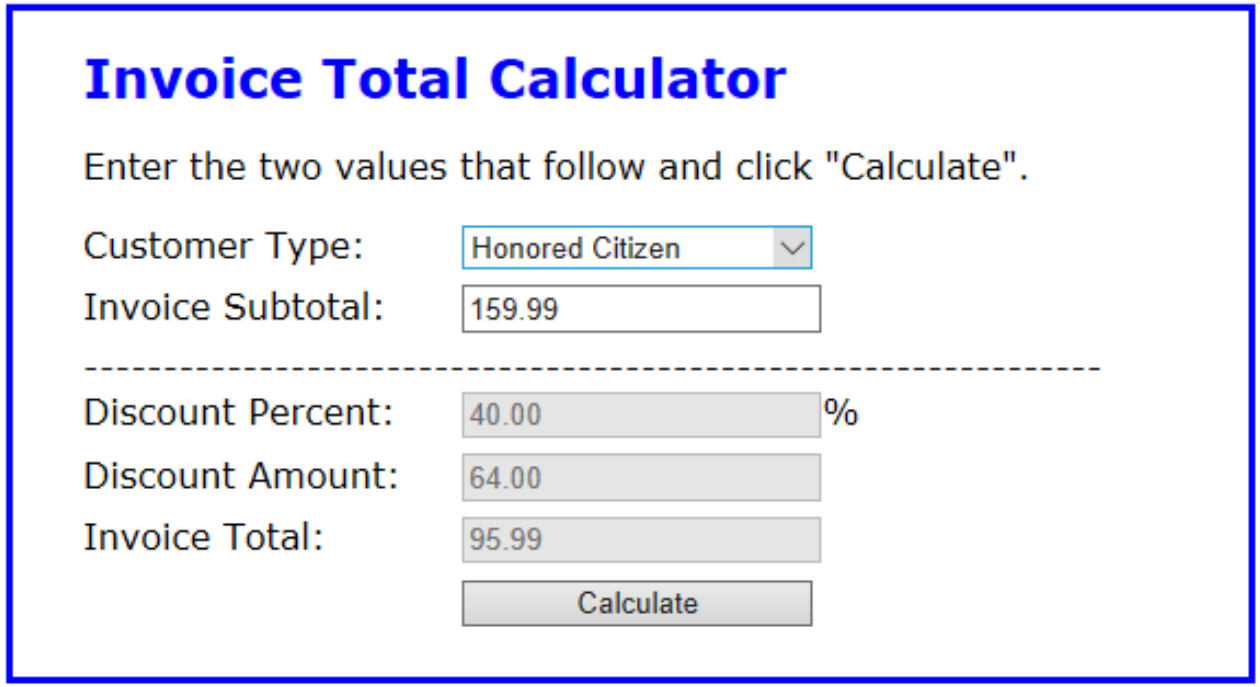
  - Example 4:

```
var selected =dt;
if(!selected){
    selected = new Date();
}
```

The Invoice Application

The Invoice Application

- The User Interface



The Invoice Application (cont.)

- The HTML code

```
<main>
    <h1>Invoice Total Calculator</h1>
    <p>Enter the two values that follow and click "Calculate".</p>

    <label for="type">Customer Type:</label>
    <select id="type">
        <option value="reg">Regular</option>
        <option value="loyal">Loyalty Program</option>
        <option value="honored">Honored Citizen</option>
    </select>
    <br>

    <label for="subtotal">Invoice Subtotal:</label>
    <input type="text" id="subtotal" /><br>
    ----------------------------------------------------------------<br>

    <label for="percent">Discount Percent:</label>
    <input type="text" id="percent" disabled />%<br>

            <label for="discount">Discount Amount:</label>
    <input type="text" id="discount" disabled /><br>

            <label for="total">Invoice Total:</label>
    <input type="text" id="total" disabled /><br>

    <label> </label>
    <input type="button" id="calculate" value="Calculate" />
</main>
```

The Invoice Application (cont.)

- The JavaScript code

```
$( document ).ready(function() {
    var calculateDiscount = function(customer, subtotal) {
        switch( customer ) {
            case "reg":
                if (subtotal < 100) {
                    return 0;
                } else if (subtotal >= 100 && subtotal < 250) {
                    return .1;
                } else if (subtotal >= 250 && subtotal < 500) {
                    return .25;
                } else if (subtotal >= 500) {
                    return .3;
                }
                break;
            case "loyal":
                return .3;
                break;
            case "honored":
                return (subtotal < 500) ? .4 : .5;
                break;
            default:
                return 0;
                break;
        }
    };
```

The Invoice Application (cont.)

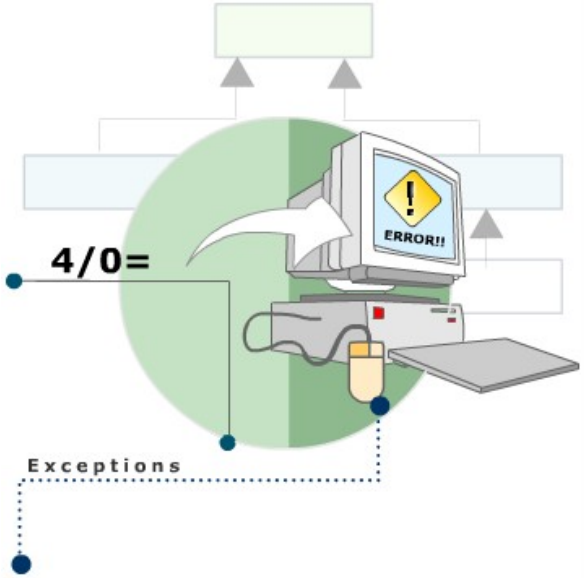
- The JavaScript code

```
$("#calculate").click(function() {
    var discountAmount, invoiceTotal, discountPercent;

    // get values from page
    var customerType = $("#type").val();
    var subtotal = $("#subtotal").val() || 0;  // default value of zero
    subtotal = parseFloat(subtotal);

    // call function to get discount percent
    discountPercent = calculateDiscount(customerType, subtotal);

    // calculate discount amount and invoice total
    discountAmount = subtotal * discountPercent;
    invoiceTotal = subtotal - discountAmount;

    // display subtotal to 2 decimals, and all other values
    $("#subtotal").val( subtotal.toFixed(2) );
    $("#percent").val( (discountPercent * 100).toFixed(2) );
    $("#discount").val( discountAmount.toFixed(2) );
    $("#total").val(  invoiceTotal.toFixed(2) );

    // set focus on type drop-down
    $("#type").focus();
});

// set focus on on type drop-down on initial load
$("#type").focus();
});
```

How to handle exceptions

Concept of Exception

- An exception is an abnormal condition that arise out of an extraordinary situation disrupting the flow of program's instruction
- In programs, exceptions can occur due to any of the followi reasons:
  - Programming errors
  - Client code errors
  - Errors beyond the control of a program



Handle checked exception

- Two ways handle checked exception:
  - Throw the exception to the calling method
  - Catch the exception and handle it

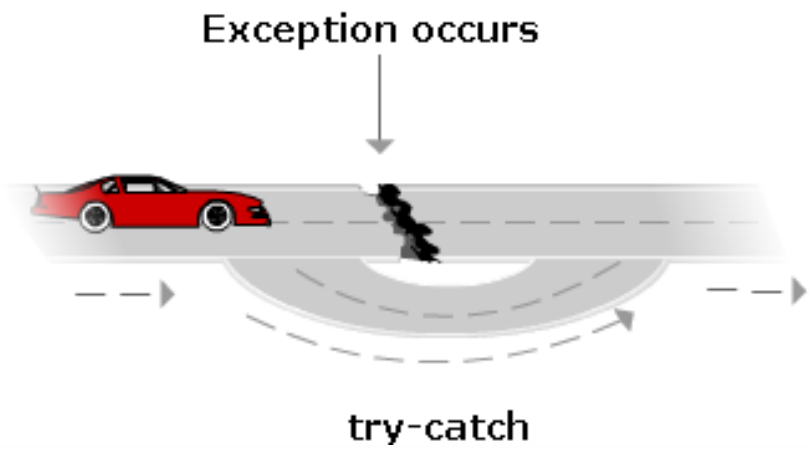
Why you need to handle an exception?

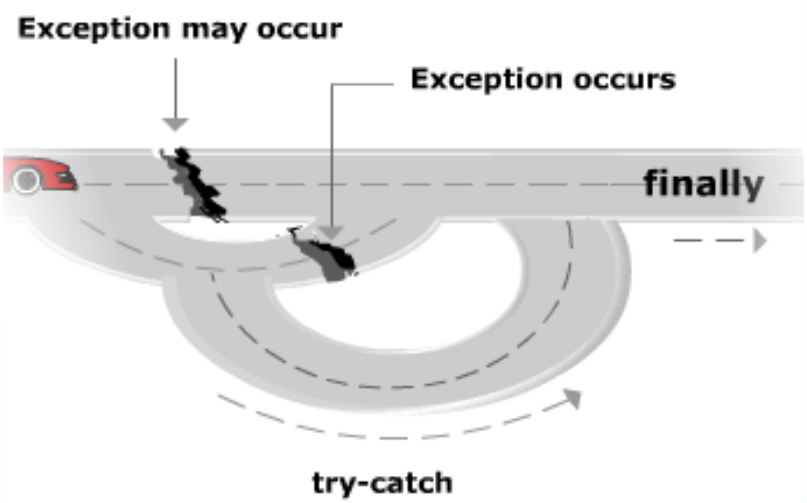
- If you don’t handle an exception:
  - Program maybe crash
  - User may be lost their data
  - Some resource can’t reuse

How to use the try statement

- The syntax of the try statement

```
try {statements}
catch(errorName) {statements}
[finally {statements}]
```





How to use the try statement(cont.)

- The properties of Error object

Property	Description
name	The type of error
message	The message that describes the error.

- A try-catch statement for a calculateFV() function

```
Var calculateFV = function(investment,rate,year){
   try{
    var futureValue = investment;
    for(var i=1; i<=years; i++){
        futureValue +=futureValue * Rate/100;
    }
    futureValue = futureValue.toFixed(2);
    return futureValue;
   }catch(error){
    alert(error.name + “ : ” + error.message);
   }
};
```

How to create and throw Error objects

- In some case, you can throw your own exceptions.
- To throw an exception you need to create an error object and use throw statement.

- **The syntax for creating a new Error object**

```
new Error(message)
```

- **The syntax for throw statement**

```
throw errorObject;
```

How to create and throw Error objects (cont.)

- A calculateFV() method that throws a new Error object

```
Var calculateFV = function(investment,rate,year){
   if(isNaN(investment) || investment <=0){
    throw new Error(“Investment must be greater than 0”);
   }
   if(isNaN(rate) || rate <=0){
    throw new Error(“Annual rate must be greater than 0”);
   }
   var futureValue = investment;
   for(var i=1; i<years; i++){
    futureValue += futureValue * rate/100;
   }
   return futureValue.toFixed(2);
};
```

How to create and throw Error objects (cont.)

- A try-catch statement that catches the Error object that has been thrown

```
try{
   $(“fucture_value”).text =
      calculateFV(investment,rate,years);
}catch(error){
   alert(error.name + “:” + error.message);
}finally{
   $(“investment”.focus())
};
```

How to create and throw Error objects (cont.)

- Some of the error types in the Error hierarchy

Type	Thrown when
`RangeError`	A numeric value has exceeded the allowable range.
`ReferenceError`	A variable is read that hasn't been defined.
`SyntaxError`	A runtime syntax error is encountered
`TypeError`	The type of a value is difference from what was expected.

- A statement that throws a RangeError object

```
throw new RangeError("Annual rate is invalid.");
```

How to use regular expressions

How to create and use regular expression

- A ***regular expression*** defines a ***pattern*** that can be searched for in a string.
- Two ways to create a regular expression object
  - By using the RegExp() constructor

```
var variableName =
    new RegExp("expression"[,"flags"]);
```

  - By coding a regular expression literal

```
var variableName = /expression/["flags"];
```

- One method of regular expression

Method	Description
`test(string)`	Searches for the regular expression in the string. It is return true if pattern is found and false if it is not found.

How to create and use regular expression (cont.)

- Two statements that create a regular expression

```
var pattern = new RegExp(“Babbage”);
var pattern = /Babbage/;
```

- How to use test() method of a regular expression
  - Two string to test

```
var inventor = “Charls Babbage”;
var programmer = “Ada Lovelace”;
```

  - How to use the test() method to search for the pattern

```
alert(pattern.test(inventor));  //display true
alert(pattern.test(programmer));//display false
```

How to match special characters and types of characters

- How to match special characters

Pattern	Matches
`\\`	Backslash character
`\/`	Forward slash
`\t`	Tab
`\n`	New line
`\r`	Carriage return
`\f`	Form feed
`\v`	Vertical tab
`[\b]`	Backspace
`\udddd`	The Unicode character whose value s the four hexadecimal digits
`\xdd`	The Latin-1 character whose value is the two hexadecimal digits

How to match special characters and types of characters (cont.)

- Examples

```
var string =“©2017 MMA Inc.
      \nAll rights reserved (5/2017).”;
alert(/\//.test(string));
          //Matches / and display true
alert(/\xA9/.test(string));
          //Matches © and display true
var pattern = new RegExp(“\\\\”); //same as /\\/
alert(pattern.test(string));
      //displays false since there ‘s no \
```

How to match special characters and types of characters (cont)

- How to match types of characters

Pattern	Matches
`.`	Any character except a newline
`[ ]`	Any character in the brackets
`[^]`	Any character not in the brackets
`[a-z]`	Any character in the range of characters
`\w`	Any letter, number or underscore
`\W`	Any character that’s not a letter, number or underscore
`\d`	Any digit
`\D`	Any character that’s not a digit.
`\s`	Any whitespace character
`\S`	Any character that’s not a whitespace

How to match special characters and types of characters (cont.)

- Examples

```
var string =“The product code is MBT-3461.”;
alert(/MB./.test(string));    //display true
alert(/MB[TF]/.test(string));    //display true
alert(/MBT-\W/.test(string));    //display false
```

How to match string positions, subpatterns, and repeating patterns

- How to match string positions

Pattern	Matches
^	The beginning of the string
$	The end of the string
\b	Word characters that aren’t followed or preceded by a word character.
\B	Word characters that are followed or preceded by a word character.

How to match string positions, subpatterns, and repeating patterns (cont.)

- Examples

```
var inventor = “Charles Babbage”;
alert(/^Charles/.test(inventor));   //display true
alert(/Babbage$/.test(inventor));   //display true
alert(/^Babbage/.test(inventor));   //display false

var programmer =“Ada Lovelace”;
alert(/Ad/.test(programmer));    //display true
alert(/Ad\b/.test(programmer));  //display false
```

How to match string positions, subpatterns, and repeating patterns (cont.)

- How to group and match subpatterns

Pattern	Matches
`(subpattern)`	Create a subpattern
`|`	Matches either the left or right subpattern
`\n`	Matches the subpattern in the specificed position

- Examples

```
var name = “Rob Robertson”;
alert(/^(Rob)|(Bob)\b/.test(name)); //display true
alert(/^(\w\w\w)\1/.test(name)); //display true
```

How to match string positions, subpatterns, and repeating patterns (cont.)

- How to match a repeating pattern

Pattern	Matches
`{n}`	Pattern must repeat exactly n times
`{n,}`	Pattern must repeat n or more times
`{n,m}`	Subpattern must repeat from n to m times
?	Zero or one of the previous subpattern(same as {0,1})
+	Zero or more of the previous subpattern(same as {1,})
*	Zero or one of the previous subpattern(same as {0,})

How to match string positions, subpatterns, and repeating patterns (cont.)

- Examples

```
var phone = “559-555-6627”;
var fax = “(559) 555 6635”;
alert(/^\d{3}-\d{4}$/.test(phone)); //display true
alert(/^\(\d{3}\)?\d{3}-\d{4}$/.test(fax));
            //display true

var phonePattern=/^(\d{3}-)|(\(\d{3}\)?)\d{3}-\d{4}$/;
alert(phonePattern.test(phone)); //display true
alert(phonePattern.test(fax));   //display true
```

Regular expressions for data validation

- Regular expressions for testing validity
  - A pattern for testing phone numbers in this format: 999-999-9999

```
/^\d{3}-\d{3}-\d{4}$/
```

  - A pattern for testing credit card number in this format: 9999-9999-9999-9999

```
/^\d{4}-\d{4}-\d{4}-\d{4}$/
```

  - A pattern for testing zip codes in either of these formats: 99999 or 99999-9999

```
/^\d{5}(-\d{4})?$/
```

Regular expressions for data validation (cont.)

- Regular expressions for testing validity
  - A pattern for testing email in this format: username@mailserver.domainname

```
/^[\w\.\-] + @[\w\.\-] + \.[a-zA-Z]+$/
```

  - A pattern for testing dates in this format: mm/dd/yyyy

```
/^[0,1]?\d\/[0-3]\d\/\d{4}$/
```

Regular expressions for data validation (cont.)

- Examples that use these expressions
  - Testing a phone number for validity

```
var phone =“559-555-6624”; //Valid phone number
var phonePattern =/^\d{3}-\d{3}-\d{4}$/
if(!phonePattern.test(phone)){
   alert(“Invalid phone number”);
}
```

  - Testing a date for valid format, but not for a valid month, day and year

```
var startDate =“8/10/2017”;   //Invalid date
var datePattern =/^[0,1]?\d\/[0-3]\d\/\d{4}$/
if(!datePattern.test(startDate)){
   alert(“Invalid start date”);
}
```

The Account Profile application

The Account Profile application

- The User interface

My Account Profile

E-Mail: grace@yahoo Please enter a valid email.

Mobile phone: 555-123-456 Please enter a phone number in NNN-NNN-NNNN format.

ZIP Code: 1234 Please enter a valid zip code.

Date of Birth: 15/18/1980 Please enter a valid date in MM/DD/YYYY format.

Save

The Account Profile application

- The HTML Code

```
<main>
    <h1>My Account Profile</h1>
    <label for="email">E-Mail:</label>
        <input type="text" name="email" id="email">
        <span> </span><br>
    <label for="phone">Mobile phone:</label>
        <input type="text" name="phone" id="phone">
        <span> </span><br>
    <label for="zip">ZIP Code:</label>
        <input type="text" name="zip" id="zip">
        <span> </span><br>
    <label for="dob">Date of Birth:</label>
        <input type="text" name="dob" id="dob">
        <span> </span><br>

    <input type="button" id="save" value="Save">
</main>
```

The Account Profile application

- The JavaScript Code

```
$(document).ready(function(){
    var isDate = function(date) {
        if ( ! /^[01]?\d\/[0-3]\d\/\d{4}$/.test(date) )  { return false; }

        var index1 = date.indexOf( "/" );
        var index2 = date.indexOf( "/", index1 + 1 );
        var month = parseInt( date.substring( 0, index1 ) );
        var day = parseInt( date.substring( index1 + 1, index2 ) );

        if ( month < 1 || month > 12 ) { return false; }
        if ( day > 31 ) { return false; }
        return true;
    };

    $( "#save" ).click(function() {
        $("span").text("");   // clear any previous error messages
        var isValid = true;   // initialize isValid flag

        var email = $("#email").val();
        var phone = $("#phone").val();
        var zip = $("#zip").val();
        var dob = $("#dob").val();
```

The Account Profile application

- The JavaScript Code

```
    if ( email === "" ||
            ! /^[\w\.\-]+@[\w\.\-]+\.[a-zA-Z]+$/.test(email) )
    {
        isValid = false;
        $( "#email" ).next().text("Please enter a valid email.");
    }
    if ( phone === "" || ! /^\d{3}-\d{3}-\d{4}$/.test(phone) ) {
        isValid = false;
        $( "#phone" ).next().text(
            "Please enter a phone number in NNN-NNN-NNNN format.");
    }
    if ( zip === "" || ! /^\d{5}(-\d{4})?$/.test(zip) ) {
        isValid = false;
        $( "#zip" ).next().text("Please enter a valid zip code.");
    }
    if ( dob === "" || !isDate(dob) ) {
        isValid = false;
        $( "#dob" ).next().text(
            "Please enter a valid date in MM/DD/YYYY format.");
    }

    if ( isValid ) {
        // code that saves profile info goes here
    }

    $("#email").focus();
});

// set focus on initial load
$("#email").focus();
});
```

Summary

- The ***equality operators*** perform type ***coercion*** whenever that's necessary. That means they convert data from one type to another before perform operator.
- The ***identity operators*** do not perform type coercion. If the two operands are different types, the result will always be false.
- An exception is an abnormal condition that arise out of an extraordinary situation disrupting the flow of program's instruction.
- You can use try/catch statement to handle an exeception.
- A ***regular expression*** defines a ***pattern*** that can be searched for in a string.

The End.