Chapter 11
Conditional Execution
so far, all of the code you have written has been *unconditionally executed*

- the browser carried out statements in the same set order

In contrast, many programming tasks require code that reacts differently under varying circumstances or conditions

- e.g., a student's course grade depends upon his/her average
- e.g., an ESP test requires recognizing when a subject guessed right
- e.g., the outcome of a game depends upon die rolls or player moves

*conditional execution* refers to a program’s ability to execute a statement or sequence of statements only if some condition holds true
If Statements

In JavaScript, the simplest form of conditional statement is the *if* statement:

- one action is taken if some condition is true, but a different action is taken if the condition is not true (called the *else case*).
- The else case is optional.

General form of the if statement:

```javascript
if (BOOLEAN_TEST) {
    STATEMENTS_EXECUTED_IF_TRUE
} else {
    STATEMENTS_EXECUTED_IF_FALSE
}
```
Braces in If Statements

some people prefer braces on separate lines formatted like this:

```c
if (BOOLEAN_TEST)
{
    STATEMENTS_EXECUTED_IF_TRUE
}
else
{
    STATEMENTS_EXECUTED_IF_FALSE
}
```

either style is acceptable, but be consistent!
- properly aligning the code (with if-else lining up and statements indented) is central in producing code that is easy to read and modify
- technically, you can omit the braces if there is only one statement
  - however, THIS IS STRONGLY DISCOURAGED!
  - can lead to tricky errors if the code is ever modified
Boolean Tests

the test that controls an if statement can be any boolean expression (i.e., an expression that evaluates to either true or false)

- boolean tests are formed using relational operators because they test the relationships between values

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>Comparison Defined by the Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
</tbody>
</table>

NOTE:

== is for comparisons
= is for assignments

the boolean test in an if statement determines the code that will be executed

- if the test is true, then the code inside the subsequent curly braces will execute
- if the test is false, then the code inside the curly braces following the else will execute
- note that if the test is false and there is no else case, the program moves on to the statement directly after the if
If Statement Examples

an if statement is known as a control statement, since its purpose is to control the execution of other statements
<html>
<!-- ifdemo.html
Dave Reed -->
<!-- This program warns a student of a failing grade. -->
<!-- =================================== -->

<head>
<title> If Demo Page </title>
<script type="text/javascript">
  function showMessage()
  // Assumes: gradeBox contains a grade
  // Results: displays a warning in response to a failing grade
  {
    var grade;
    grade = document.getElementById("gradeBox").value;
    grade = parseFloat(grade);
    if (grade < 60) {
      alert("You failed! Time to hit the books.");
    }
  }
</script>
</head>

<body>
<p> Your grade: <input type="text" id="gradeBox" size="10" value="" />
</p>
<p> </p>
<input type="button" value="click for Message" onclick="ShowMessage();o" />
</p>
</body>
</html>
recall that values entered via text boxes/areas are always returned as strings

```javascript
if (document.getElementById('age').value >= 18) {
    alert("You are old enough to vote.");
} else {
    alert("Sorry. You are too young to vote.");
}
```

will say that a 2-year old can vote, but a 102-year old can't!

WHY?

if you wish to treat a value obtained from a text box or text area as a number, you must use the `parseFloat` function to convert it

```javascript
age = parseFloat(document.getElementById('age').value);
if (age >= 18) {
    alert("You are old enough to vote.");
} else {
    alert("Sorry. You are too young to vote.");
}
```

will behave as expected
Nested If Statements

programming tasks often require code that responds to more than one condition

- this can be accomplished by nesting one if statement inside of another

example: determining wind-chill

- wind-chill is only defined for temperatures less than or equal to 50 degrees
- the initial if test is to determine if it is a valid temperature to calculate wind-chill
- the nested if statement only executes if the outer test is true

```javascript
if (temperature <= 50) {
    if (windSpeed <= 3) {
        windChill = temperature;
    } else {
        windChill = 35.74 + 0.6215*temperature + (0.4275*temperature - 35.75)*Math.pow(windSpeed, 0.16);
    }
} else {
    alert("Wind-chill is defined only if temperature <= 50.");
    windChill = NaN;
}
```
nested if-else structures are known as *cascading if-else statements* because control cascades down the branches

- the topmost level is evaluated first
- if the test succeeds, then the corresponding statements are executed and control moves to the next statement following the cascading if
- if the test fails, then control cascades down to the next if test
- in general, control cascades down the statement from one test to another until one succeeds or the end of the statement is reached

**example: nested if-else structure**

```javascript
if (grade < 60) {
    alert("You failed! Time to hit the books.");
} else {
    if (grade < 90) {
        alert("You passed, but could do better.");
    } else {
        alert("Congratulations! You got an A.");
    }
}
```
A Cleaner Notation

when it is necessary to handle a large number of alternatives, nested if-else statements can become cumbersome and unwieldy

- multiple levels of indentation and curly braces cause the code to look cluttered
- make it harder to read/understand

example:      

```java
if (grade < 60) {
    letterGrade = "F";
} else {
    if (grade < 70) {
        letterGrade = "D";
    } else {
        if (grade < 90) {
            letterGrade = "C";
        } else {
            letterGrade = "A";
        }
    }
}
```

vs. a more readable else-if

```java
if (grade < 60) {
    letterGrade = "F";
} else if (grade < 70) {
    letterGrade = "D";
} else if (grade < 80) {
    letterGrade = "C";
} else if (grade < 90) {
    letterGrade = "B";
} else {
    letterGrade = "A";
}
```
consider a Web page that simulates the roll of a single die

- will use an image to display the die
- will use a button to initiate the die roll

- when the user clicks the button, a random die roll is selected and the corresponding image is displayed
1. <html>
2.  <!-- dice.html -->
3.  <!-- This page simulates and displays the roll of a die. -->
4.  <!-- -->
5. 
6.  <head>
7.  <title> Die Rolls </title>
8.  <script type="text/javascript"
10.  </script>
11.  <script type="text/javascript">
12.     function Roll() {
13.         // Assumes: die images are in dave-reed.com/book/Images
14.         // Results: displays a randomly selected image of a 6-sided die
15.         
16.         var roll;
17.         
18.         roll = RandomInt(1, 6);
19.         
20.         if (roll == 1) {
22.         }
23.         else if (roll == 2) {
25.         }  
26.         else if (roll == 3) {
28.         }
29.         else if (roll == 4) {
31.         }
32.         else if (roll == 5) {
34.         }  
35.         else {  
37.         }  
38.     }
39.  </script>
40.  </head>
41.  <body style="text-align:center">
42.      <p>
43.         <img id="die" alt="die image"
45.      </p>
46.  </body>
47.  </html>

Die Roll Page

the RandomInt function from random.js is used to select the random roll depending on the roll value, the correct image is displayed since more than two possibilities, a cascading if-else is needed
note that each case in the cascading if-else follows the same pattern

```javascript
if (roll == 1) {
} else if (roll == 2) {
} else if (roll == 3) {
} else if (roll == 4) {
} else if (roll == 5) {
} else {
}
```

this entire cascading if-else structure could be replaced by the following:

```javascript
```
Counters

in software applications, if statements are often used to count occurrences of conditional or user-initiated events
- e.g., count the number of times dice rolls come up doubles
- e.g., count the number of times the user guesses a number correctly

any variable that is used to record occurrences of an event is known as a counter
- initially, the counter is set to zero
- each time the specified action occurs, the counter is incremented
- after a given time period, the value stored in the counter will tell you the number of times the desired event took place

```javascript
document.getElementById("numRolls").value = 
parseFloat(document.getElementById("numRolls").value) + 1;
```
Logical Connectives

sometimes, simple comparisons between two values may not be adequate to
express the conditions under which code should execute

JavaScript provides operators for expressing multipart tests

- logical AND (&&): represents the conjunction of two things
  - (TEST1 && TEST2) is true if both TEST1 and TEST2 are true

```javascript
if (roll1 == 4 && roll2 == 4) {
    // code to be executed when double fours are rolled
}
```

- logical OR (||): represents the disjunction of two things
  - (TEST1 || TEST2) is true if either TEST1 or TEST2 are true

```javascript
if (roll1 == 4 || roll2 == 4) {
    // code to be executed when at least one four is rolled
}
```

- logical NOT (!): represents negation
  - (!TEST1) is true only if TEST1 is false

```javascript
if (!(roll1 == 4 || roll2 == 4)) {
    // code to be executed when neither roll is a four
}
```