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:

```cpp
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
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
Accessing Text Fields

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
If Statement Examples

```javascript
if (grade < 60) {
    alert('You failed! Time to hit the books.');}  // code executed if grade < 60
}

if (grade < 60) {
    diff = 60 - grade;
    alert('You failed! If only you could have ' +
        'earned ' + diff + ' more points.');
}  // code executed if grade < 60

if (grade < 60) {
    diff = 60 - grade;
    alert('You failed! If only you could have ' +
        'earned ' + diff + ' more points.'
}  // code executed if grade < 60
else {
    alert('Congratulations, you passed.');  // code executed otherwise (grade >= 60)
}
```

An if statement is known as a control statement, since its purpose is to control the execution of other statements.
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: three different grade levels
- failing (grade < 60), acceptable (60 ≤ grade < 90), A-level (grade ≥ 90)
- the outer if-else distinguishes failing from passing grades
- the nested if-else further separates passing grades into acceptable and A-level

```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.');
    }
}
```
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

```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.');
}
```
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 and make it harder to read/understand
- can simplify by removing some unnecessary curly braces & aligning each case to the left

example:

```plaintext
nested if statements     vs.     more readable else-if

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';
            }
        }
    }
}

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 two dice

- will use image to display the dice
- will use a button to initiate the die rolls
- will keep track and display the number of rolls

- when the user clicks the button, two random die rolls are selected, the corresponding images are displayed, and the number of rolls incremented
Counters

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('rollSpan').innerHTML =
parseFloat(document.getElementById('rollSpan').innerHTML)+1;
```

in software applications, counters are often conditional

- 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

conditional counters must be controlled by if statements

- if the desired event occurs, then you increment the counter

```javascript
if (roll1 == roll2) {
  // CODE TO BE EXECUTED WHEN DOUBLES ARE ROLLED
}
```
Stats Page

The RandomInt function from random.js is used to select the random roll since each die image is stored as die#.gif, can assign each image source with one assignment the number of rolls appears in a span

- initially 0
- incremented in RollDice each time the button is clicked
Boolean Expressions

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 DOUBE 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
  }
  ```
Slot Machine

initially, displays three random slot images at the click of a button

need to display player's credits
- player starts with 20 credits
- each spin costs 1 credit, three matching slots earns 13 credits
- disallow play if no credits
- possibly give a loan when broke

```html
<!doctype html>
<html>
<head>
  <title>Slot Machine</title>
  <script type="text/javascript" src="http://balance3e.com/random.js"></script>
  <script type="text/javascript">
    function SpinSlots()
    // Assumes: slot images are in http://balance3e.com/Images
    // Results: displays 3 random slot images
    {
      var slot1, slot2, slot3;
      slot1 = RandomOneOf(['lemon', 'cherry', 'bar', 'donut']);
      slot2 = RandomOneOf(['lemon', 'cherry', 'bar', 'donut']);
      slot3 = RandomOneOf(['lemon', 'cherry', 'bar', 'donut']);
      document.getElementById('slot1Img').src = 'http://balance3e.com/Images/' + slot1 + '.jpg';
      document.getElementById('slot2Img').src = 'http://balance3e.com/Images/' + slot2 + '.jpg';
      document.getElementById('slot3Img').src = 'http://balance3e.com/Images/' + slot3 + '.jpg';
    }
  </script>
</head>
<body>
  <div style="text-align:center">
    <p>
      <img id="slot1Img" border=1 alt="slot image" src="http://balance3e.com/Images/cherry.jpg">
      <img id="slot2Img" border=1 alt="slot image" src="http://balance3e.com/Images/lemon.jpg">
      <img id="slot3Img" border=1 alt="slot image" src="http://balance3e.com/Images/bar.jpg">
    </p>
    <input type="button" value="Click to Spin" onclick="SpinSlots();">
  </div>
</body>
</html>
```