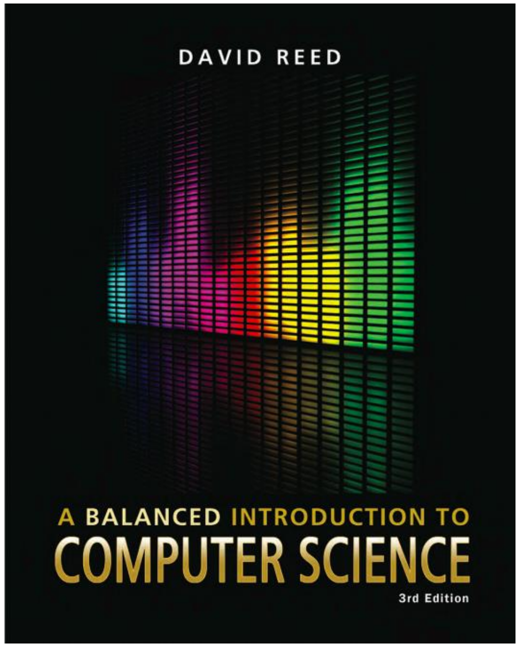
A Balanced Introduction to Computer Science, 3/E

David Reed, Creighton University

©2011 Pearson Prentice Hall
ISBN 978-0-13-216675-1



Chapter 13
Conditional Repetition

Conditional Repetition

an if statement is known as a *control statement*

- it is used to control the execution of other JavaScript statements

- provides for conditional execution
- is useful for solving problems that involve choices
  - *either do this or don't, based on some condition* (if)
  - *either do this or do that, based on some condition* (if-else)

closely related to the concept of conditional execution is *conditional repetition*

- many problems involve repeating some task over and over until a specific condition is met

- e.g., rolling dice until a 7 is obtained
- e.g., repeatedly prompting the user for a valid input

- in JavaScript, *while loops* provide for conditional repetition

While Loops

a *while loop* resembles an if statement in that its behavior is dependent on a Boolean condition.

- however, the statements inside a while loop's curly braces (a.k.a. the *loop body*) are executed *repeatedly* as long as the condition remains true
- general form:

```
while (BOOLEAN_TEST) {
    STATEMENTS_EXECUTED_AS_LONG_AS_TRUE
}
```

when the browser encounters a while loop, it first evaluates the Boolean test

- if the test succeeds, then the statements inside the loop are executed in order, *just like an if statement*
- once all the statements have been executed, program control returns to the beginning of the loop
- the loop test is evaluated again, and if it succeeds, the loop body statements are executed *again*
- this process repeats until the Boolean test fails

While Loop Example

example: roll two dice repeatedly until doubles are obtained

```
roll1 = RandomInt(1, 6);                  // ROLL AND DISPLAY DICE
roll2 = RandomInt(1, 6);
document.getElementById('outputDiv').innerHTML=roll1+'-'+roll2+'<br>';

while (roll1 ! = roll2) {                 // WHILE NOT DOUBLES,
  roll1 = RandomInt(1, 6);                // ROLL AGAIN AND DISPLAY AT
  roll2 = RandomInt(1, 6);                // THE END OF THE PAGE DIVISION
  document.getElementById('outputDiv').innerHTML =
      document.getElementById('outputDiv').innerHTML+
      roll1+'-'+roll2+'<br>';
}
```

sample output:

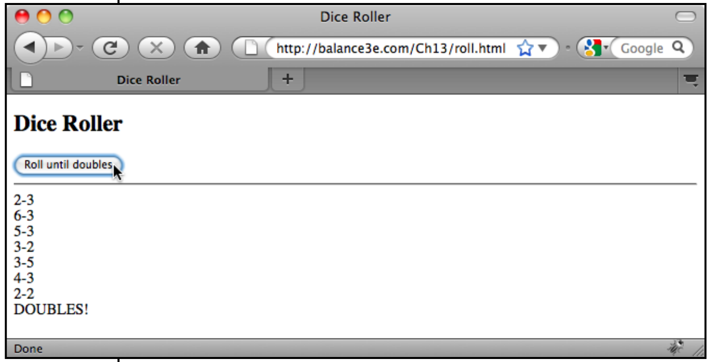
```
2-3
6-3
5-3
3-2
3-5
4-3
2-2
DOUBLES!
```

note: even though while loops and if statements look similar, they are very different control statements

- an *if statement* may execute its code once or not at all
- a *while loop* may execute its code an arbitrary number of times (including not at all)

Dice Roller Page

```
<!doctype html>
<!-- roll.html                                          Dave Reed -->
<!-- This page simulates dice rolls until doubles are obtained. -->
<!-- ========================================================== -->

<html>
 <head>
   <title> Dice Roller </title>
   <script type="text/javascript" src="http://balance3e.com/random.js">
   </script>
   <script type="text/javascript">
     function RollUntilDoubles()
     // Assumes: outputDiv is available for output
     // Results: rolls and displays dice until doubles are obtained
     {
       var roll1, roll2;

       roll1 = RandomInt(1, 6);                 // ROLL AND DISPLAY DICE
       roll2 = RandomInt(1, 6);
       document.getElementById('outputDiv').innerHTML=roll1+'-'+roll2+'<br>';

       while (roll1 ! = roll2) {                // WHILE NOT DOUBLES,
         roll1 = RandomInt(1, 6);               // ROLL AGAIN AND DISPLAY AT
         roll2 = RandomInt(1, 6);               // THE END OF THE PAGE DIVISION
         document.getElementById('outputDiv').innerHTML =
             document.getElementById('outputDiv').innerHTML+
             roll1+'-'+roll2+'<br>';
       }

       document.getElementById('outputDiv').innerHTML =
           document.getElementById('outputDiv').innerHTML+'DOUBLES!';
     }
   </script>
 </head>

 <body>
   <h2>Dice Roller</h2>
   <input type="button" value="Roll until doubles"
          onclick="RollUntilDoubles();">
   <hr>
   <div id="outputDiv"></div>
 </body>
</html>
```



Dice Roller

```
2-3
6-3
5-3
3-2
3-5
4-3
2-2
DOUBLES!
```

Avoiding redundancy

note the redundancy in the code

- must perform the initial dice roll before the loop begins
- then, have to repeatedly re-roll inside the loop

can avoid this by either:

- "priming the loop" with default values that allow the loop to execute
- defining a Boolean "flag" to determine when the loop should continue

```
roll1 = -1;                              // PRIME THE LOOP BY ASSIGNING
roll2 = -2;                              // INITIAL VALUES TO THE VARIABLES
document.getElementById('outputDiv').innerHTML = '';

while (roll1 != roll2) {                 // AS LONG AS YOU DON'T HAVE DOUBLES,
    roll1 = RandomInt(1, 6);             // ROLL AGAIN AND DISPLAY THE ROLLS
    roll2 = RandomInt(1, 6);
    document.getElementById('outputDiv').innerHTML =
        document.getElementById('outputDiv').innerHTML+roll1+'-'+roll2+'<br>';
}

____________________________________________________________________________

rolledDoubles = false;                   // INITIALIZE A BOOLEAN FLAG
document.getElementById('outputDiv').innerHTML = '';

while (rolledDoubles == false) {         // AS LONG AS IT IS FALSE,
    roll1 = RandomInt(1, 6);             // ROLL AGAIN AND DISPLAY THE ROLLS
    roll2 = RandomInt(1, 6);
    document.getElementById('outputDiv').innerHTML =
        document.getElementById('outputDiv').innerHTML+roll1+'-'+roll2+'<br>';

    if (roll1 == roll2) {                // IF DOUBLES WERE ROLLED, SET THE FLAG
        rolledDoubles = true;            // SO THAT THE LOOP WILL TERMINATE
    }
}
```

Loop Tests

note: the loop test defines the condition under which the loop continues

- this is often backwards from the way we think about loops

- e.g., read input until you get a positive number (i.e., until input > 0)

```
while (input <= 0) { . . . }
```

- e.g., keep rolling dice until you get doubles (i.e., until roll1 == roll2)

```
while (roll1 != roll2) { . . . }
```

- e.g., keep rolling dice until you get double fours (i.e., until roll1 == 4 && roll2 = 4)

```
while (roll1 != 4 || roll2 != 4) { . . . }
```

DeMorgan's Law:

```
!(X && Y) == (!X || !Y)
!(X || Y) == (!X && !Y)
```

Counter-Driven Loops

since a while loop is controlled by a condition, it is *usually* impossible to predict the number of repetitions that will occur

- e.g., how many dice rolls will it take to get doubles?

a while loop can also be used to repeat a task some fixed number of times

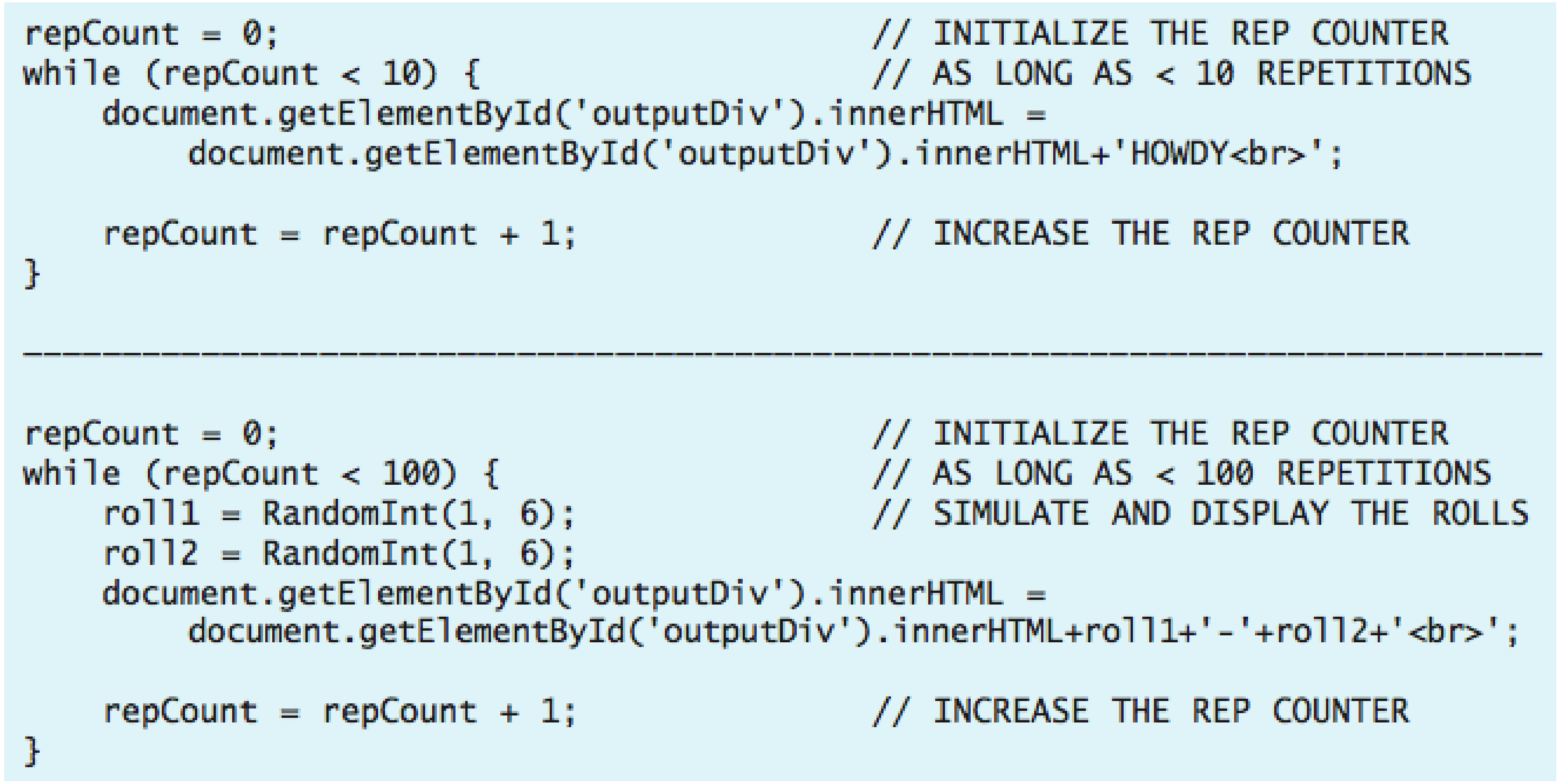
- implemented by using a while loop whose test is based on a counter
- general form of counter-driven while loop:

```
repCount = 0;
while (repCount < DESIRED_NUMBER_OF_REPETITIONS) {
    STATEMENTS_FOR_CARRYING_OUT_DESIRED_TASK
    repCount = repCount + 1;
}
```

- the counter is initially set to 0 before the loop begins, and is incremented at the end of the loop body
  - the counter keeps track of how many times the statements in the loop body have executed
  - when the number of repetitions reaches the desired number, the loop test fails and the loop terminates

Counter-Driven Loops

examples:



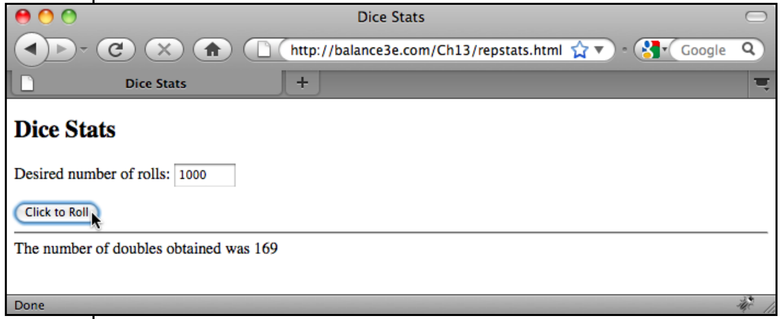
```
repCount = 0;                                   // INITIALIZE THE REP COUNTER
while (repCount < 10) {                         // AS LONG AS < 10 REPETITIONS
    document.getElementById('outputDiv').innerHTML =
        document.getElementById('outputDiv').innerHTML+'HOWDY<br>';

    repCount = repCount + 1;                    // INCREASE THE REP COUNTER
}

------------------------------------------------------------------------------

repCount = 0;                                   // INITIALIZE THE REP COUNTER
while (repCount < 100) {                        // AS LONG AS < 100 REPETITIONS
    roll1 = RandomInt(1, 6);                    // SIMULATE AND DISPLAY THE ROLLS
    roll2 = RandomInt(1, 6);
    document.getElementById('outputDiv').innerHTML =
        document.getElementById('outputDiv').innerHTML+roll1+'-'+roll2+'<br>';

    repCount = repCount + 1;                    // INCREASE THE REP COUNTER
}
```

Counter-driven Dice Roller

```
<!doctype html>
<!-- repstats.html                                          Dave Reed -->
<!-- This page simulates repeated dice rolls and maintains stats.  -->
<!-- =============================================================  -->

<html>
 <head>
   <title> Dice Stats </title>
   <script type="text/javascript" src="http://balance3e.com/random.js">
   </script>
   <script type="text/javascript">
     function RollRepeatedly()
     // Assumes: repsBox contains a non-negative integer
     // Results: simulates that many dice rolls, displays # of doubles
     {
       var totalRolls, doubleCount, repCount, roll1, roll2;

       totalRolls = parseFloat(document.getElementById('repsBox').value);

       doubleCount = 0;                        // INITIALIZE THE COUNTERS
       repCount = 0;
       while (repCount < totalRolls) {         // REPEATEDLY,
           roll1 = RandomInt(1, 6);            // SIMULATE THE DICE ROLLS
           roll2 = RandomInt(1, 6);
           if (roll1 == roll2) {               // IF DOUBLES,
               doubleCount = doubleCount + 1;  // INCREMENT THE COUNTER
           }

           repCount = repCount + 1;            // INCREMENT THE REP COUNTER
       }
                                               // DISPLAY THE RESULTS
       document.getElementById('outputDiv').innerHTML =
           'The number of doubles obtained was ' + doubleCount;
     }
   </script>
 </head>

 <body>
   <h2>Dice Stats</h2>
   <p>
    Desired number of rolls:
    <input type="text" id="repsBox" size=6 value=1000>
   </p>
   <input type="button" value="Click to Roll" onclick="RollRepeatedly();">
   <hr>
   <div id="outputDiv"></div>
 </body>
</html>
```

while loop executes totalRolls times

- each time, the dice are rolled and checked for doubles



Dice Stats

Desired number of rolls: 1000

The number of doubles obtained was 169

Infinite Loops

the browser will repeatedly execute statements in the body of a while loop as long as the loop test succeeds (evaluates to true)

- it is possible that the test will always succeed and the loop will run forever

```
repCount = 0;
while (repCount < 10) {
    document.getElementById('outputDiv').inerHTML =
      document.getElementById('outputDiv').inerHTML + 'HOWDY<br>';
}
```

- a loop that runs forever is known as an *infinite loop* (or a *black hole loop*)

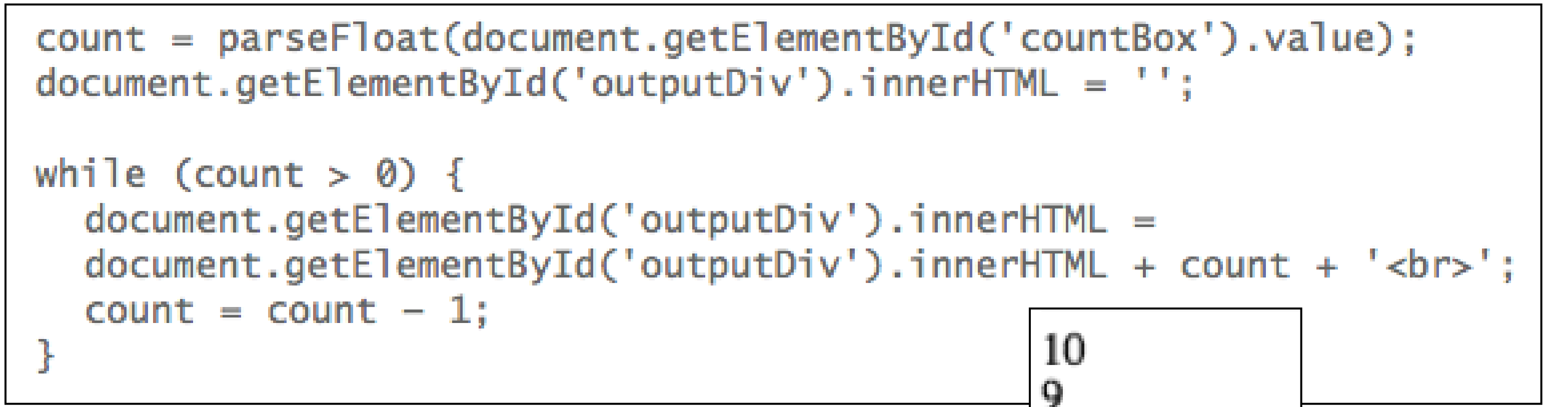
- to guard against infinite loops, make sure that some part of the loop test changes inside the loop
  - in the above example, `repCount` is not updated in the loop so there is no chance of terminating once the loop starts

- an infinite loop may freeze up the browser
  - sometimes, clicking the Stop button will suffice to interrupt the browser
  - other times, you may need to restart the browser

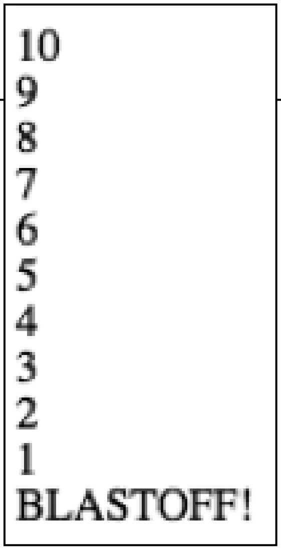
Variables and Repetition

any variable can be employed to control the number of loop repetitions and the variable can be updated in various ways

example: countdown



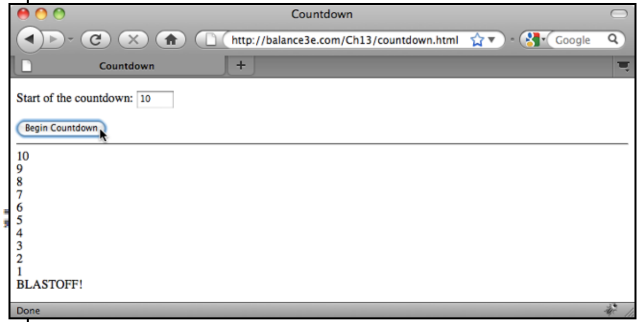
```
count = parseFloat(document.getElementById('countBox').value);
document.getElementById('outputDiv').innerHTML = '';

while (count > 0) {
   document.getElementById('outputDiv').innerHTML =
   document.getElementById('outputDiv').innerHTML + count + '<br>';
   count = count - 1;
}
```



```
10
9
8
7
6
5
4
3
2
1
BLASTOFF!
```

Countdown Page

```
<!doctype html>
<!-- countdown.html                                       Dave Reed -->
<!-- This page displays a countdown from a specified number. -->
<!-- ========================================================= -->

<html>
 <head>
   <title> Countdown </title>
   <script type="text/javascript">
     function Countdown()
     // Assumes: countBox contains a non-negative integer
     // Results: displays a countdown from that number in outputDiv
     {
       var count;

       count = parseFloat(document.getElementById('countBox').value);
       document.getElementById('outputDiv').innerHTML = '';

       while (count > 0) {
         document.getElementById('outputDiv').innerHTML =
         document.getElementById('outputDiv').innerHTML + count + '<br>';
         count = count - 1;
       }

       document.getElementById('outputDiv').innerHTML =
           document.getElementById('outputDiv').innerHTML + 'BLASTOFF!';
     }
   </script>
 </head>

 <body>
   <p>
    Start of the countdown:
    <input type="text" id="countBox" size=4 value=10>
   </p>
   <input type="button" value="Begin Countdown" onclick="Countdown();">
   <hr>
   <div id="outputDiv"></div>
 </body>
</html>
```



```
Start of the countdown: 10
```

```
10
9
8
7
6
5
4
3
2
1
BLASTOFF!
```

Example: Hailstone Sequences

an interesting unsolved problem in mathematics: hailstone sequence

1. start with any positive integer
2. if the number is odd, then multiply the number by three and add one; otherwise, divide it by two
3. repeat as many times as desired

- for example: 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, …

it is conjectured that, no matter what positive integer you start with, you will always end up in the 4-2-1 loop

- this has been verified for all starting number up to 5,764,607,523,034,234,880
- but, it still has not been proven to hold for ALL starting numbers