Week 6 Overview

- Week 5 review
  - Exam 1 (no new material)
Week 6 Overview

• Outcomes
  • Sketch the solution to a problem requiring iteration.
  • Write correct iterative code to solve a given problem.
  • Identify and correct common loop errors such as off-by-one errors, infinite loops, and non-executing loops.
Repetition

• Repetition (aka Iteration)
  • Not many problems are solvable using only straight line and conditional execution with if/else and case statements.
  • Need an additional control structure that lets us execute the same code while some condition is true.

Example:
• Input a number representing the length of a line and then “draw” the line using asterisks:
• Input: 6, output: ******
Repetition

• An effort without repetition:

```javascript
function makeLine(length, ch){
    var str = "";
    if (length >= 1)
        str += ch;
    if (length >= 2)
        str += ch;
    if (length >= 3)
        str += ch;
    //... and so on
    return str;
}
```

Since the length is variable based on the user’s input, there is no way to write enough code to handle all possible inputs!

• A better solution:

Create an empty string and set a counter to 0

Is counter < length? 

True

Append another asterisk onto the string

Increment the counter by 1

False
Repetition

• A better solution:

Create an empty string and set a counter to 0

Is counter < length?

True

Append another asterisk onto the string

Increment the counter by 1

False

This line allows us to jump backwards and do something over again! Called a “loop.”

Repetition

• A better solution:

• Four parts to loops

• Initialization

Create an empty string and set a counter to 0

Is counter < length?

True

Append another asterisk onto the string

Increment the counter by 1

False

Sets up the loop so that it can be executed the first time (especially the condition variables). Ex: counter=0;
Repetition

• A better solution:
• Four parts to loops
  • Initialization
  • Condition

A better solution: Create an empty string and set a counter to 0.

Four parts to loops:

• Initialization
• Condition
• Body

Is counter < length?

- True: Append another asterisk onto the string.
- False: Increment the counter by 1.

A Boolean expression that controls whether or not the loop executes again. Ex: counter < length

The statement(s) that need to be repeatedly executed in order to solve the given problem. Ex: str += "*";
Repetition

• A better solution:
• Four parts to loops
  • Initialization
  • Condition
  • Body
  • Update

A change to one of the condition variables that makes progress toward the condition becoming false. Ex: ++counter;

Just like with if/else and case statements, there is one path in and one path out of the control structure.
Repetition

• Try it yourself
  • Suppose `makeLine(length, ch)` exists, and produces a string of the given length using the given character.
  • Use `makeLine` to create a loop that produces a right triangle of a given height:
    
    ```
    *
    **
    ***
    ****
    *****
    
    Here, the height is 5.
    ```
Repetition

- Pre-test loops
  - Condition is evaluated before the body of the loop is executed.
  - Key idea: body may not ever execute.

- Post-test loops
  - Condition is evaluated after the body of the loop is executed.
  - Key idea: body always executes at least once
while loops

- While loops:
  - Pre-test loop syntax

```java
while (condition) {
    body_statements;
}
```

All that is really required. But, which of the four parts are missing?
while loops

- While loops:
  - Pre-test loop syntax

```
initialization;
while (condition) {
    body_statements;
    update_statement;
}
```

Create an empty string and set a counter to 0

Is counter < length?

- True
  - Append another asterisk onto the string
  - Increment the counter by 1
- False
while loops

Create an empty string and set a counter to 0

Is counter < length?

True

False

Append another asterisk onto the string

Increment the counter by 1

function makeLine(length, ch) {
  var str = "";
  var count = 0;

  while (count < length) {
    str += ch;
    ++count;
  }

  return str;
}

while loops

Create an empty string and set a counter to 0

Is counter < length?

True

False

Append another asterisk onto the string

Increment the counter by 1

Initialize

Condition

Body

Update
while loops

- Try it yourself:
  - Write the function `makeTriangle(height, ch)` that will produce the string containing a right-triangle.

```javascript
function makeTriangle(height, ch) {
    // Your code here
}
```
while loops

- A little harder problem:
  
  * Write a function `isPrime(number)` that determines if the given number parameter is prime (i.e. is only divisible evenly by 1 and itself).
  
  * Hint 1: loop through all the numbers `[2...(number-1)]`
  
  * Hint 2: if the remainder when dividing is zero (modulus), then it is not prime.

```plaintext
Counter < number?
Counter ≤ number: False

number % counter == 0?
Counter == 0?: True

Return false

Increment the counter by 1
```
Set a counter to 2

counter < number?

True

number % counter == 0?

False

Return false

Increment the counter by 1

function isPrime(number) {
    var counter = 2;
    while (counter < number) {
        if (number % counter == 0) {
            return false;
        }
        ++counter;
    }
    return true;
}

Which is harder?

This is writing code (i.e. translating algorithms into source code).

This is program design (i.e. problem solving using algorithms).
for loops

• For loops:
  • Pre-test loop syntax

```java
for (initialization; condition; update) {
    body_statements;
}
```

This is precisely equivalent to:

```java
initialization;
while (condition) {
    body_statements;
    update;
}
```
for loops

- Rewriting isPrime using a for-loop

```javascript
function isPrime(number) {
    for (var counter = 0; counter < number; ++counter) {
        if (number % counter == 0) {
            return false
        }
    }
    return true;
}
```

for vs. while loops

- When to use for vs. while
  - Equivalent at runtime
  - while loops are a little more flexible (i.e. the update step can be conditional or in the middle of the body)
  - for loops are generally used for counting (i.e. the bounds are known)
do...while loops

- do...while loops:
- Post-test loop syntax

```java
do {
    body_statements;
} while (condition);
```

initialization;
```java
do {
    body_statements;
    update;
} while (condition);
```

Required elements.

All 4 elements.
do...while loops

- Post-test loops
  - Body always guaranteed to execute at least once.
  - But, we could still copy-and-paste the body above a pre-test loop and achieve the same results.

Example: read input using `prompt()` ensuring that the user enters a positive number

```javascript
var number;
do {
    number = parseInt(prompt("Enter a positive number"));
} while (isNaN(number) || number < 0);
alert("Read number: " + number);
```

Key idea: initialization and update use the same code, so a natural fit for do...while
Case study

**Investment Calculator**

*Author: Todd A. Whitaker*

This program calculates the future value of an ongoing investment (i.e., an annual contribution to a retirement account) using compounding interest.

**Inputs:**
- Annual contribution: 600
- Number of years: 5
- Interest rate: 4.5%

**Output:**

<table>
<thead>
<tr>
<th>Year</th>
<th>Principle</th>
<th>Interest Paid</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$600.00</td>
<td>$27.00</td>
<td>$627.00</td>
</tr>
<tr>
<td>2</td>
<td>$1,227.00</td>
<td>$55.21</td>
<td>$1,282.21</td>
</tr>
<tr>
<td>3</td>
<td>$1,882.21</td>
<td>$84.70</td>
<td>$1,966.91</td>
</tr>
<tr>
<td>4</td>
<td>$2,566.91</td>
<td>$115.51</td>
<td>$2,682.43</td>
</tr>
<tr>
<td>5</td>
<td>$3,282.13</td>
<td>$117.71</td>
<td>$3,399.84</td>
</tr>
</tbody>
</table>

Total contribution: $3,000.00
Total interest: $430.13
Case Study

• Set up the HTML page

```html
<html>
<head>
  <title>Investment Calculator</title>
  <script type="text/javascript" src="div.js"></script>
  <script type="text/javascript" src="get.js"></script>
  <script type="text/javascript" src="Investments.js"></script>
</head>
<body>
  <h1>Investment Calculator</h1>
  <p><em>Author: Todd A. Whittaker</em></p>
  <fieldset><legend>Inputs:</legend>
    <table>
      <tr>
        <td><label for="contrib">Annual contribution:</label></td>
        <td><input type="text" id="contrib" value="500" /></td>
      </tr>
      <tr>
        <td><label for="years">Number of years:</label></td>
        <td><input type="text" id="years" value="20" /></td>
      </tr>
      <tr>
        <td><label for="interest">Interest rate:</label></td>
        <td><input type="text" id="interest" value="4.5" /></td>
      </tr>
    </table>
  </fieldset>
  <input type="button" value="Execute!" onclick="main('contrib', 'years', 'interest', 'output')" />
</body>
</html>
```
Case Study

• Set up the HTML page

```html
<fieldset>
  <legend>Output:</legend>
  <div id="output"></div>
</fieldset>
</body>
</html>
```

• Input validation

```javascript
function main(contribID, yearsID, interestID, outputID) {
  var contrib = getFloat(contribID);
  var years = getInt(yearsID);
  var interest = getFloat(interestID);

  var errors = "";
  if (isNaN(contrib) || contrib < 0) {
    errors += "Contribution has a bad value.\n"
  }
}
```
Case Study

• Input validation

```javascript
if (isNaN(years) || years < 0) {
    errors += "Years has a bad value.\n"
}
if (isNaN(interest) || interest < 0) {
    errors += "Interest rate has a bad value.\n"
}
if (errors != "") {
    alert(errors);
    return;
}
var tableHtml = calculate(contrib, years, interest);
setDiv(outputID, tableHtml);
```

• Calculations

```javascript
function calculate(contrib, years, interest) {
    var sumPrinciple = 0;
    var sumInterest = 0;
    var result = "";

    result += "<table border='1'><tr><th>Year</th>" +
              "<th>Principle</th><th>Interest Paid</th>" +
              "<th>Total</th></tr>";
```
**Case Study**

**Calculations**

```javascript
for (var year = 1; year <= years; ++year) {
    sumPrinciple += contrib;
    var interest = sumPrinciple * (interestRate / 100);
    result += '<tr>";
    result += td(year);
    result += td("$" + sumPrinciple.toFixed(2));
    result += td("$" + interest.toFixed(2));
    result += td("$" + (sumPrinciple + interest).toFixed(2));
    result += '</tr>";
    sumPrinciple += interest;
}
```

**function td(val) {**
  `return '<td>' + val + '</td>';`**}**
Case Study

• Calculations

```javascript
result += "</table><br />";
result += "Total contribution: $" +
  (contrib * years).toFixed(2) + "\n\n";
result += "Total interest: $" +
  (sumPrinciple - contrib * years).toFixed(2) + "\n\n";
return result;
```

ITEC 136
Business Programming Concepts

Week 06, Part 05
Common Loop Errors

Franklin University
FOUNDED 1902
Common Loop Errors

- Loop errors are due to problems with each of the four components.
  - Initialization or Condition
    - Could cause the loop to never execute

```javascript
var i = 10;
while (i < 10) {
  document.writeln("i is " + i + "<br />");
  ++i;
}
```

i starts at 10, and 10 is not less than 10, so the loop body never executes.
Common Loop Errors

• Loop errors are due to problems with each of the four components.
  • Initialization or Condition
    • Could execute one too many times

```javascript
var i = 0;
while (i <= 10) {
    document.writeln("i is " + i + "<br />");
    ++i;
}
```

Loop body actually executes 11 times, not 10 times.
Common Loop Errors

- Loop errors are due to problems with each of the four components.
  - Initialization or Condition
    - Could execute one too few times

```javascript
var i = 1;
while (i < 10) {
    document.writeln("i is "+i+"<br />");
    ++i;
}
```

Loop body actually executes 9 times, not 10 times.
Common Loop Errors

• Loop errors are due to problems with each of the four components.
  • Condition or update
    • Could execute forever

Tip: Programmers start counting with 0 and use < as their condition. To execute 10 times, initialize “i” to 0 and set the condition to “i<10”. This is typical of left-bound-included and right-bound-excluded.

Remember 2\textsuperscript{nd} grade number lines: [a, b)

```
var i = 1;
while (i < 10) {
    document.writeln("i is " + i + "<br />");
    ++i;
}
```

Loop body actually executes 9 times, not 10 times.

```
var i = 0;
while (i < 10) {
    document.writeln("i is " + i + "<br />");
}
```
Common Loop Errors

- Loop errors are due to problems with each of the four components.
  - Condition or update
    - Could execute \textit{forever}

```javascript
var i = 0;
while (i < 10) {
    document.writeln("i is \" + i + \"<br />\")
}
```

Note that the update step is \textit{missing}. No progress is made toward the condition being false. An \textit{incorrect} update (i.e. "--i") would also do this.

Questions?
Self Quiz

• Name and describe the four basic parts of every loop.

• Compare and contrast pre-test vs. post-test loops. What Javascript constructs correspond to each?

• Name 3 common loop errors and where to look for bugs.
Self Quiz

• What is the output of the following code segment?

```javascript
function whatAmI(number)
{
    var x = 0, y = 1;
    for (var i = 0; i < number; ++i) {
        var next = x + y;
        x = y;
        y = next;
    }
    return y;
}
document.writeln(whatAmI(5));
```

Self Quiz

• Rewrite the function `whatAmI` using a while-loop instead of a for-loop.

• Write a function that prints out all the numbers in the range \([a, b)\) that are evenly divisible by 7 but not divisible by 5.
Self Quiz

• Write a function that receives a parameter X, and then reads in X numbers. It should then print out the average, minimum, and maximum or the numbers read.
Upcoming Deadlines

- Homework 5 – Due February 16
- Pre-class 7 – Due February 16
- Lab 2 – Due February 23