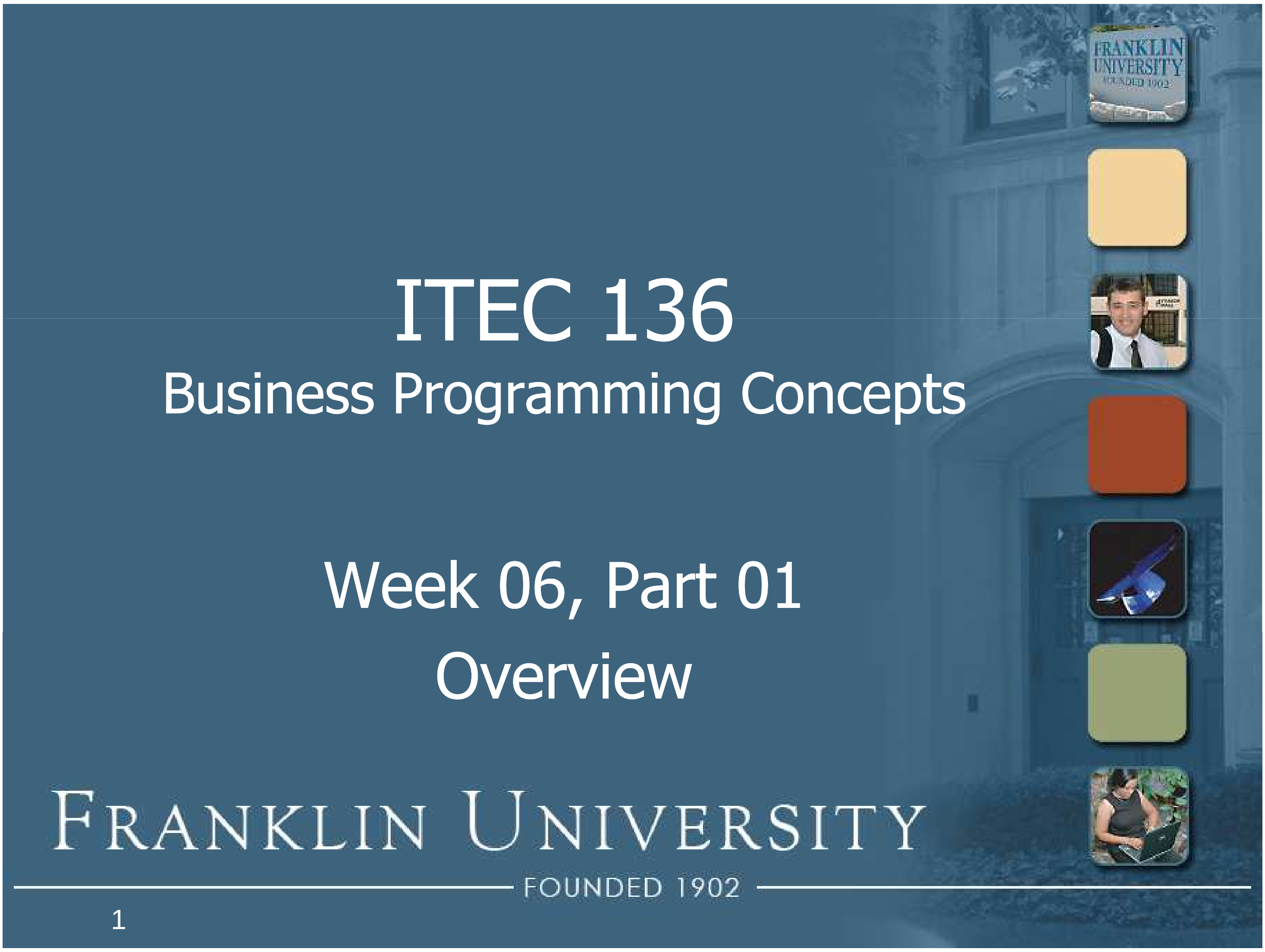


ITEC 136
Business Programming Concepts

Week 06, Part 01

Overview

FRANKLIN UNIVERSITY

FOUNDED 1902

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.

FRANKLIN UNIVERSITY

ITEC 136

Business Programming Concepts

Week 06, Part 02

Repetition

FRANKLIN UNIVERSITY

FOUNDED 1902

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.

Repetition

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

Repetition

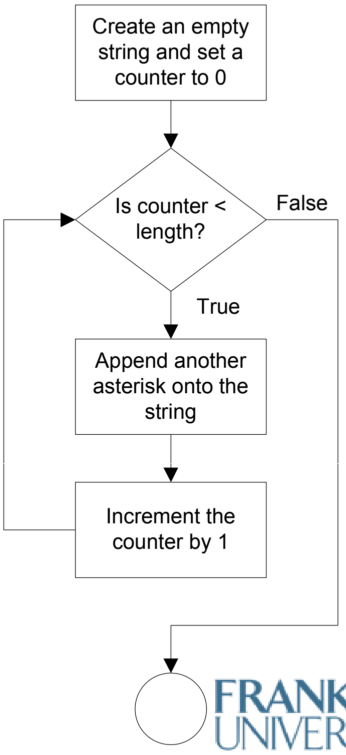
- An effort without repetition:

```
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!

Repetiton

- A better solution:



Create an empty string and set a counter to 0

Is counter < length?

False

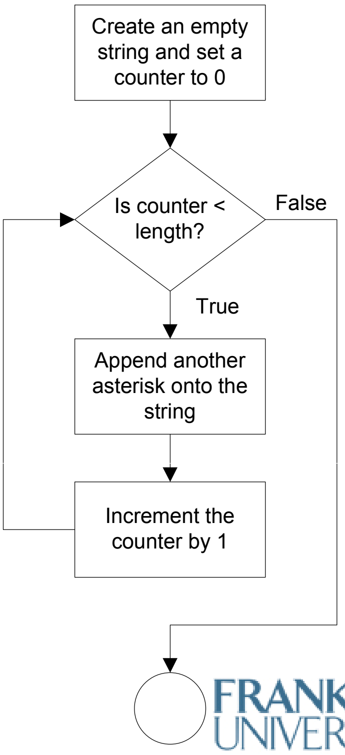
True

Append another asterisk onto the string

Increment the counter by 1

Repetiton

- A better solution:



Create an empty string and set a counter to 0

Is counter < length?

True

False

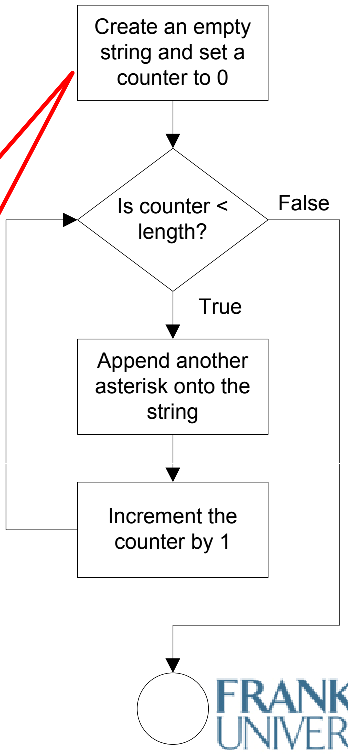
Append another asterisk onto the string

Increment the counter by 1

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

Repetiton

- A better solution:
- Four parts to loops
  - Initialization



Create an empty string and set a counter to 0

Is counter < length?

True

False

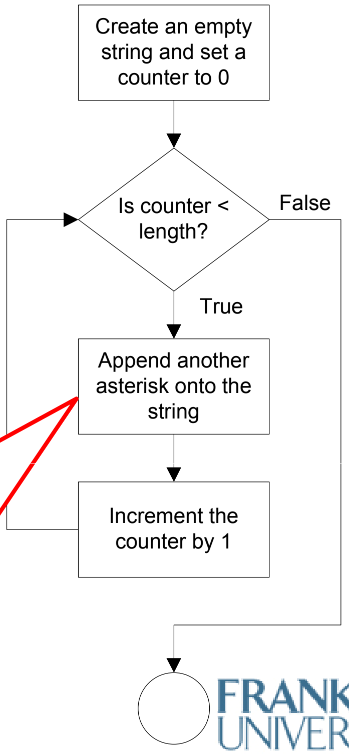
Append another asterisk onto the string

Increment the counter by 1

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

Repetiton

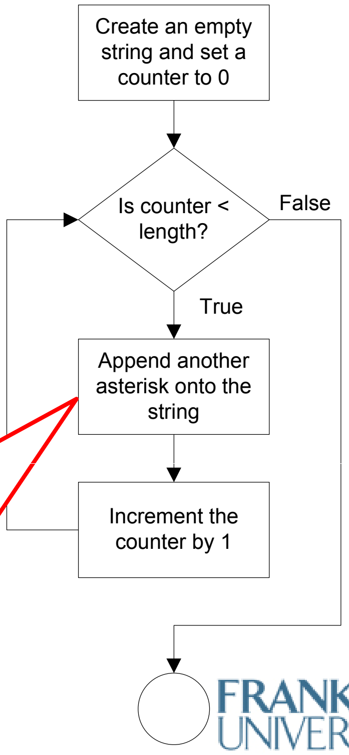
- A better solution:
- Four parts to loops
  - Initialization
  - Condition


Create an empty string and set a counter to 0
Is counter < length?
False
True
Append another asterisk onto the string
Increment the counter by 1

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

Repetiton

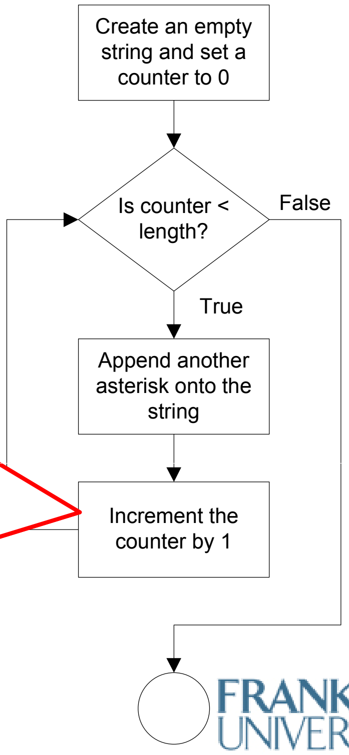
- A better solution:
- Four parts to loops
  - Initialization
  - Condition
  - Body


Create an empty string and set a counter to 0
Is counter < length?
False
True
Append another asterisk onto the string
Increment the counter by 1

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

Repetiton

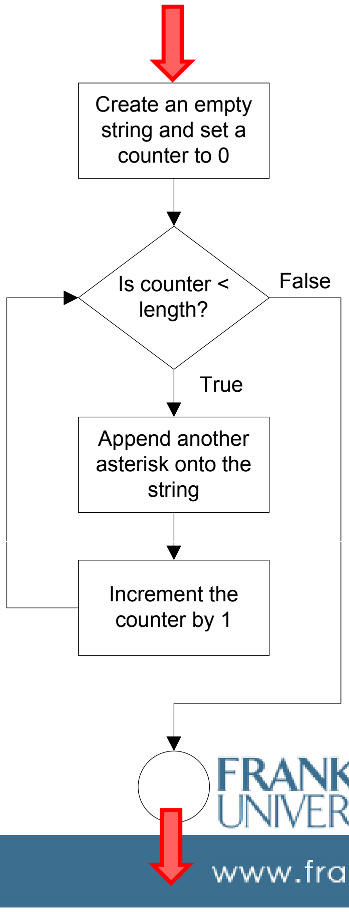
- A better solution:
- Four parts to loops
  - Initialization
  - Condition
  - Body
  - Update


Create an empty string and set a counter to 0
Is counter < length?
False
True
Append another asterisk onto the string
Increment the counter by 1

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

Repetiton

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


Create an empty string and set a counter to 0
Is counter < length?
False
True
Append another asterisk onto the string
Increment the counter by 1

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

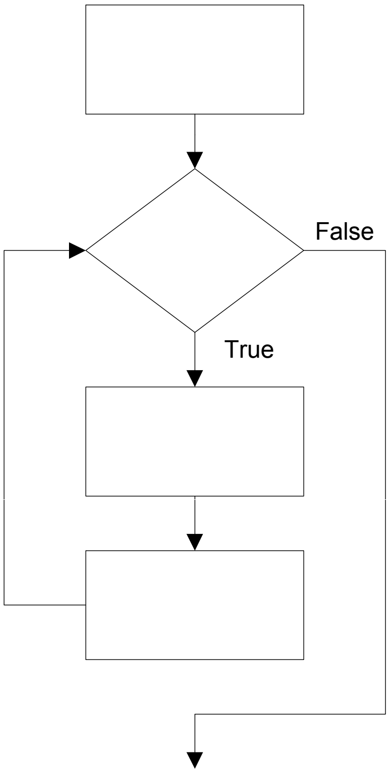
Repetition

- 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:

```
*
**
***
****
*****
```

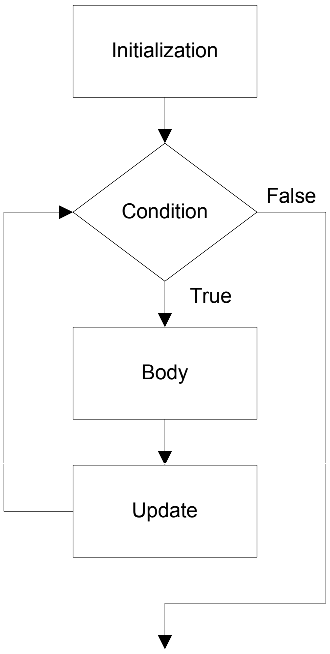
Here, the height is 5.

Repetition



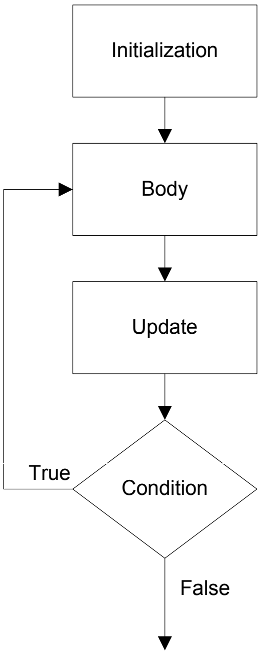
Repetition

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



Repetition

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



ITEC 136

Business Programming Concepts

Week 06, Part 03

Loop structure syntax

FRANKLIN UNIVERSITY

FOUNDED 1902

while loops

- While loops:
  - Pre-test loop syntax

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

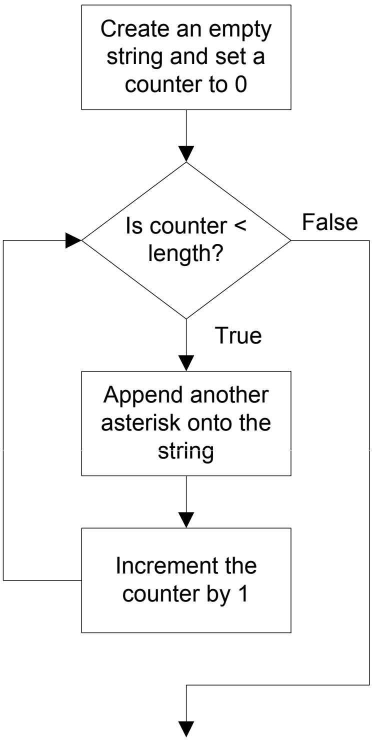
All that is really required. But, which of the four parts are missing?

while loops

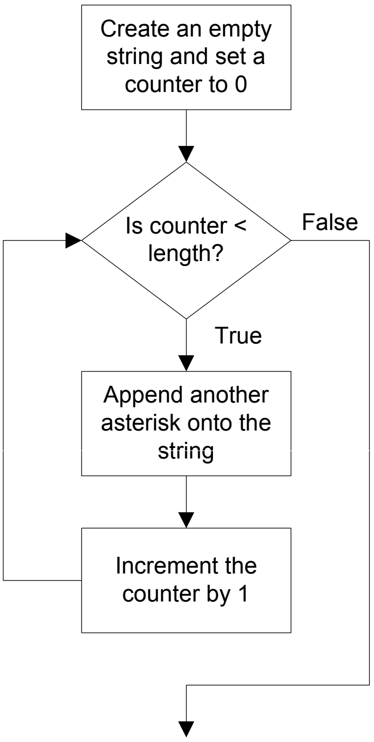
- While loops:
  - Pre-test loop syntax

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

while loops

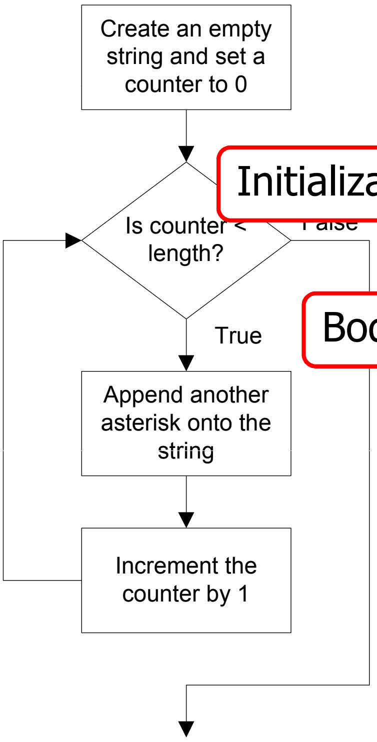

Create an empty string and set a counter to 0
Is counter < length?
False
True
Append another asterisk onto the string
Increment the counter by 1

while loops


Create an empty string and set a counter to 0
Is counter < length?
False
True
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?
False
True
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;
}
```

Initialization

Body

Condition

Update

while loops

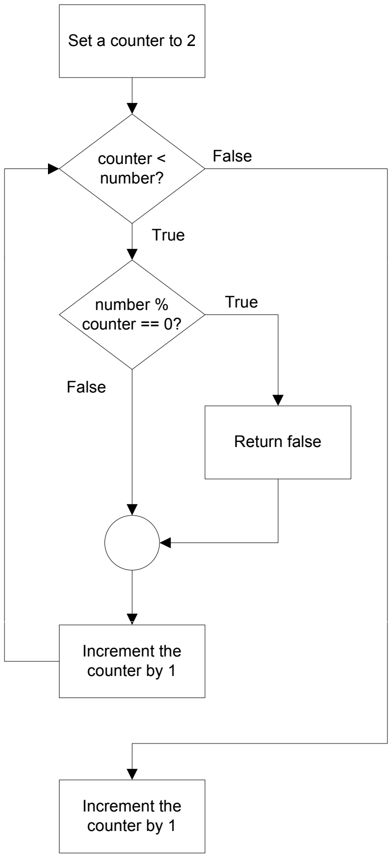
- Try it yourself:
  - Write the function `makeTriangle(height, ch)` that will produce the string containing a right-triangle.

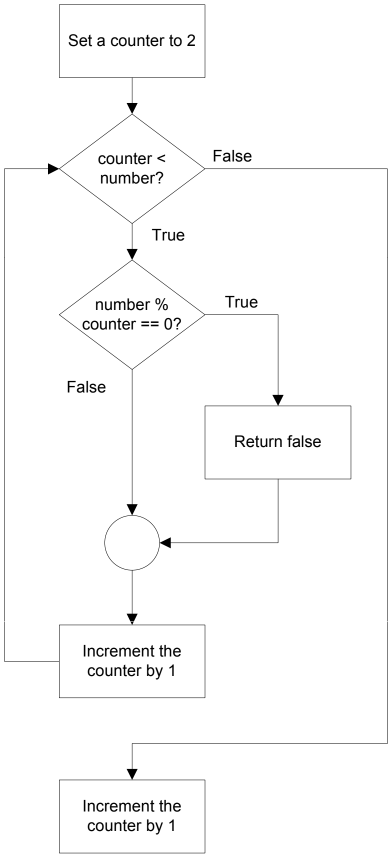
while loops

```
function makeTriangle(height, ch) {








}
```

`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.





Set a counter to 2

counter < number?

False

True

number % counter == 0?

True

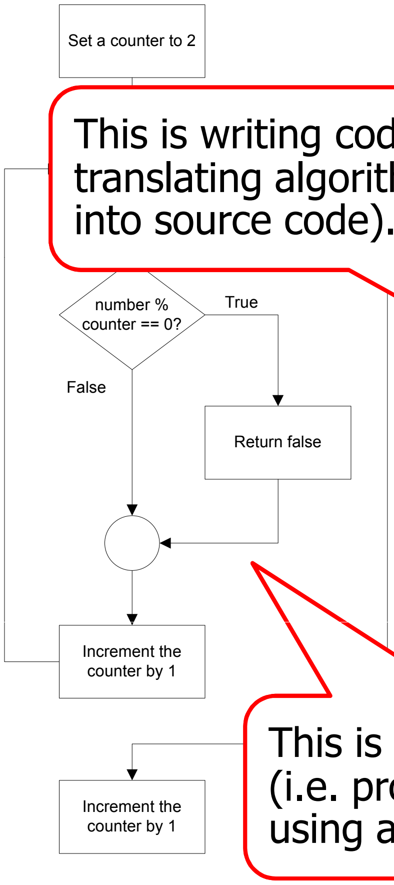
False

Return false

Increment the counter by 1

Increment the counter by 1

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



Set a counter to 2

number % counter == 0?

True

False

Return false

Increment the counter by 1

Increment the counter by 1

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

```
   ction isPrime(number){
   ar counter = 2;
  while (counter < number) {
    if (number % counter == 0) {
      return false;
    }
    ++counter
  }
  return true
}
```

Which is harder?

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

for loops

- For loops:
  - Pre-test loop syntax

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

for loops

- For loops:
  - Pre-test loop syntax

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

This is precisely equivalent to:

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

for loops

- Rewriting `isPrime` using a for-loop

```
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

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

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

do…while loops

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

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

```
initialization;
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.

do…while loops

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

```
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

ITEC 136

Business Programming Concepts

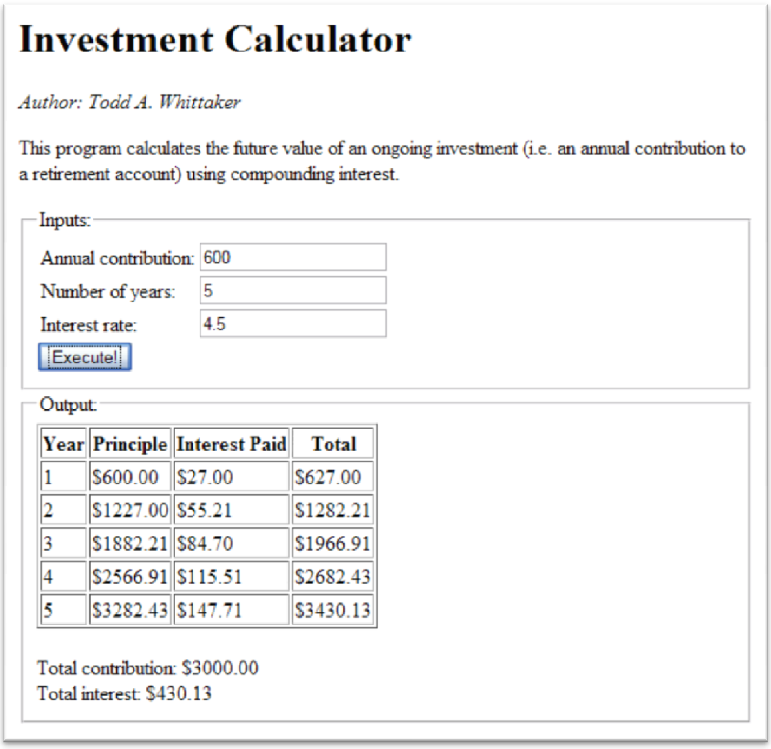
Week 06, Part 04

Case study: Investment Calculator

FRANKLIN UNIVERSITY

FOUNDED 1902

Case study



Investment Calculator

Author: Todd A. Whittaker

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:

Year	Principle	Interest Paid	Total
1	$600.00	$27.00	$627.00
2	$1227.00	$55.21	$1282.21
3	$1882.21	$84.70	$1966.91
4	$2566.91	$115.51	$2682.43
5	$3282.43	$147.71	$3430.13

Total contribution: $3000.00
Total interest: $430.13

Case Study

- Set up the HTML page

Investments.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>
```

Case Study

- Set up the HTML page

Investments.html

```
<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>
  <input type="button" value="Execute!"
    onclick="main('contrib', 'years', 'interest', 'output')" />
</fieldset>
```

Case Study

- Set up the HTML page

Investments.html

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

Case Study

- Input validation

Investments.js

```
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

Investments.js

```
    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);
}
```

Case Study

- Calculations

Investments.js

```
function calculate(contrib, years, interestRate) {
  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

Investments.js

```
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;
}
```

Case Study

- Calculations

Investments.js

```
for (var year = 1; year <= years; ++year) {
  sumPrinciple
  var interest
  result += "<t
  result += td(y
  result += td("$" +         nciple.toFixed(2));
  result += td("$" + interest.toFixed(2));
  result += td("$" + (sumPrinciple + interest).toFixed(2));
  result += "</tr>";
  sumPrinciple += interest;
}
```

```
function td(val) {
  return "<td>" + val + "</td>";
}
```

Case Study

- Calculations

Investments.js

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

  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

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

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

```
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

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

Common Loop Errors

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

```
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

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

Common Loop Errors

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

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

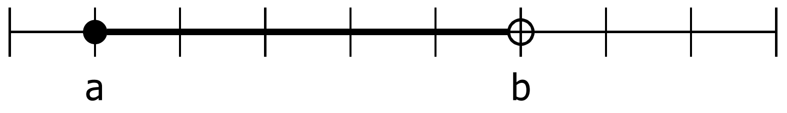
Loop body actually executes 9 times, not 10 times.

Co

- Loop er
  with ea
  - Initiali
    - Could exe ne too *few* times

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 2nd 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.

Common Loop Errors

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

```
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 *forever*

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

}
```

Note that the update step is *missing*. No progress is made toward the condition being `false`. An *incorrect* update (i.e. “`--i`”) would also do this.

Questions?

ITEC 136

Business Programming Concepts

Week 06, Part 06

Self Quiz

FRANKLIN UNIVERSITY

FOUNDED 1902

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.

FRANKLIN UNIVERSITY

Self Quiz

- What is the output of the following code segment?

```
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.

FRANKLIN UNIVERSITY

ITEC 136

Business Programming Concepts

Week 06, Part 07

Upcoming deadlines

FRANKLIN UNIVERSITY

FOUNDED 1902

Upcoming Deadlines

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