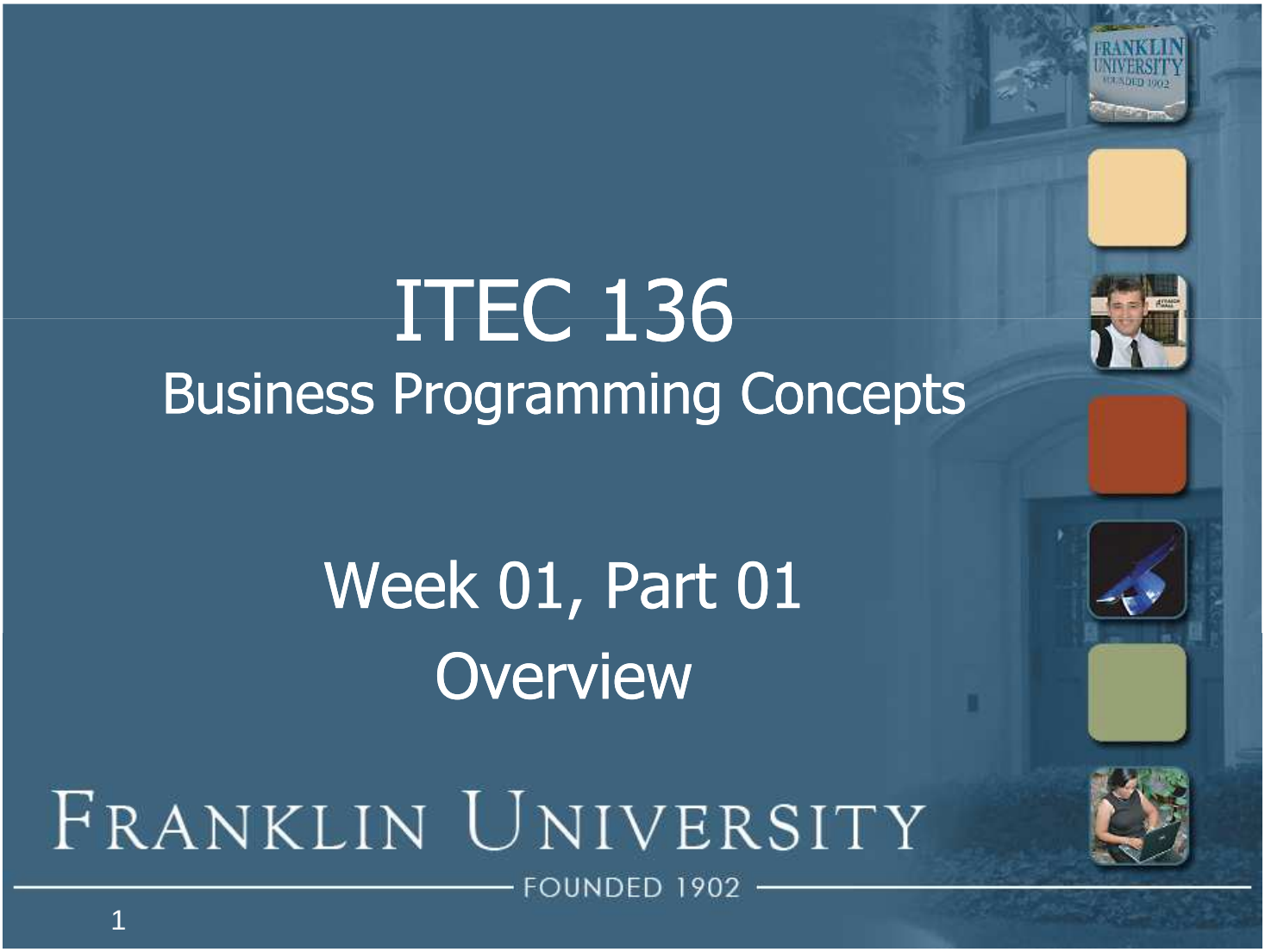


Week 7 Overview

- Week 6 review
  - Four parts to every loop
    - Initialization
    - Condition
    - Body
    - Update
  - Pre-test loops: condition is evaluated before body is executed

Week 7 Overview

- Week 6 review
  - Post-test loops: condition is evaluated after the body is executed
  - `while` loops: condition and body are explicit. Initialization and update still need to be present

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


Week 7 Overview

- Week 6 review
  - `for` loops: all four elements are explicit. Often used when bounds are explicitly known (i.e. counting loops).

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

Week 7 Overview

- Week 6 review
  - do…while loops: two elements explicit, the only post-test loop.

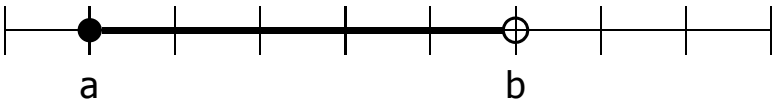
```
initialization;
do {
    body_statements;
    update;
} while (condition);
```


Week 7 Overview

- Week 6 review
  - Common loop errors
    - Off-by-one: one too many or one too few executions of the body
    - Infinite loops: never stops because the condition never becomes false
    - Body never executes: condition is false initially

Week 7 Overview

- Week 6 review
  - How programmers count
    - Always start with zero
    - Always use < as the comparison operator
    - Left bound included, right bound excluded. E.g. [a, b)



Week 7 Overview

- Outcomes
  - Implement algorithms requiring nested loops.
  - Differentiate between various loop termination conditions such as sentinels, results-controlled, symmetric and asymmetric bounds, and counting.

ITEC 136

Business Programming Concepts

Week 07, Part 02

Termination conditions

FRANKLIN UNIVERSITY

FOUNDED 1902

Termination conditions

- The condition terminates loops when it becomes false
  - Saw counting loops last week [e.g. `while (counter < max)`]
  - But, there are many different kinds of Boolean conditions.

Termination conditions

- Sentinels
  - Sentinels "guard" something, and in this case it is the end of the loop.
  - Commonly used for and end-of-data condition.

FRANKLIN UNIVERSITY
www.franklin.edu

Termination conditions

- Sentinels
  - Ex: read numbers until a non-number is entered (non-number is the sentinel)

```
function readData() {
    var data = prompt("Enter data (cancel to quit)");
    while (data != null) {
        // do somethign with the data here
        data = prompt("Enter data (cancel to quit)")
    }
}
```


UNIVERSITY
www.franklin.edu

Termination conditions

- Sentinels
  - Ex: read numbers until a non-number is entered (non-[illegible]tinel)

```
function readData() {
    var data = prompt("Enter data (cancel to quit)");
    while (data != null) {
        // do something with the data here
        data = prompt("Enter data (cancel to quit)")
    }
}
```

`null` guards the end of input (it is what `prompt` returns when the user clicks "cancel."

Termination conditions

- Sentinels
  - Any kind of data that shouldn't appear in the input stream can be a sentinel
    - A negative number
    - Zero
    - A special string

Termination conditions

- Flag controlled loops
  - Often, the termination condition can't be detected until the middle of the body.
  - Use a Boolean flag "done" set to false initially to enter the loop.
  - When the condition is detected, set done to `true`.

Termination conditions

- Flag controlled loops

```
function readData() {
  var done = false;
  while (!done) {
    var data = prompt("Enter data (cancel to quit)");
    if (data == null) {
      done = true;
    } else {
      // do something with data here
    }
  }
}
```

Termination conditions

- Flag controlled loops

```
function readData() {
  var done = false;
  while (!done) {
    var data = prompt("Enter data (cancel to quit)");
    if (data == null) {
      done = true;
    } else {
      // do something with data here
    }
  }
}
```

Set the flag so that the loop is entered initially

Termination conditions

- Flag controlled loops

```
function readData() {
  var done = false;
  while (!done) {
    var data = prompt("Ent                    ");
    if (data == null) {
      done = true;
    } else {
      // do something with data here
    }
  }
}
```

When the termination condition is detected, set the flag so that the loop will exit.

Termination conditions

- Result controlled loops
  - Body of the loop is calculating a value and we want to keep iterating until that value falls within a certain range.
  - The result of the body calculation controls the termination condition.
  - Ex: how many years of investing $10K at 5% interest to reach $1M?

Termination conditions

- Result controlled loops

```
function yearsToReach(target, principle, rate) {
    var years = 0;
    var total = 0;
    while (total < target) {
        total += principle;
        total *= (1.0 + rate)
        ++years;
    }
    return years;
}
```

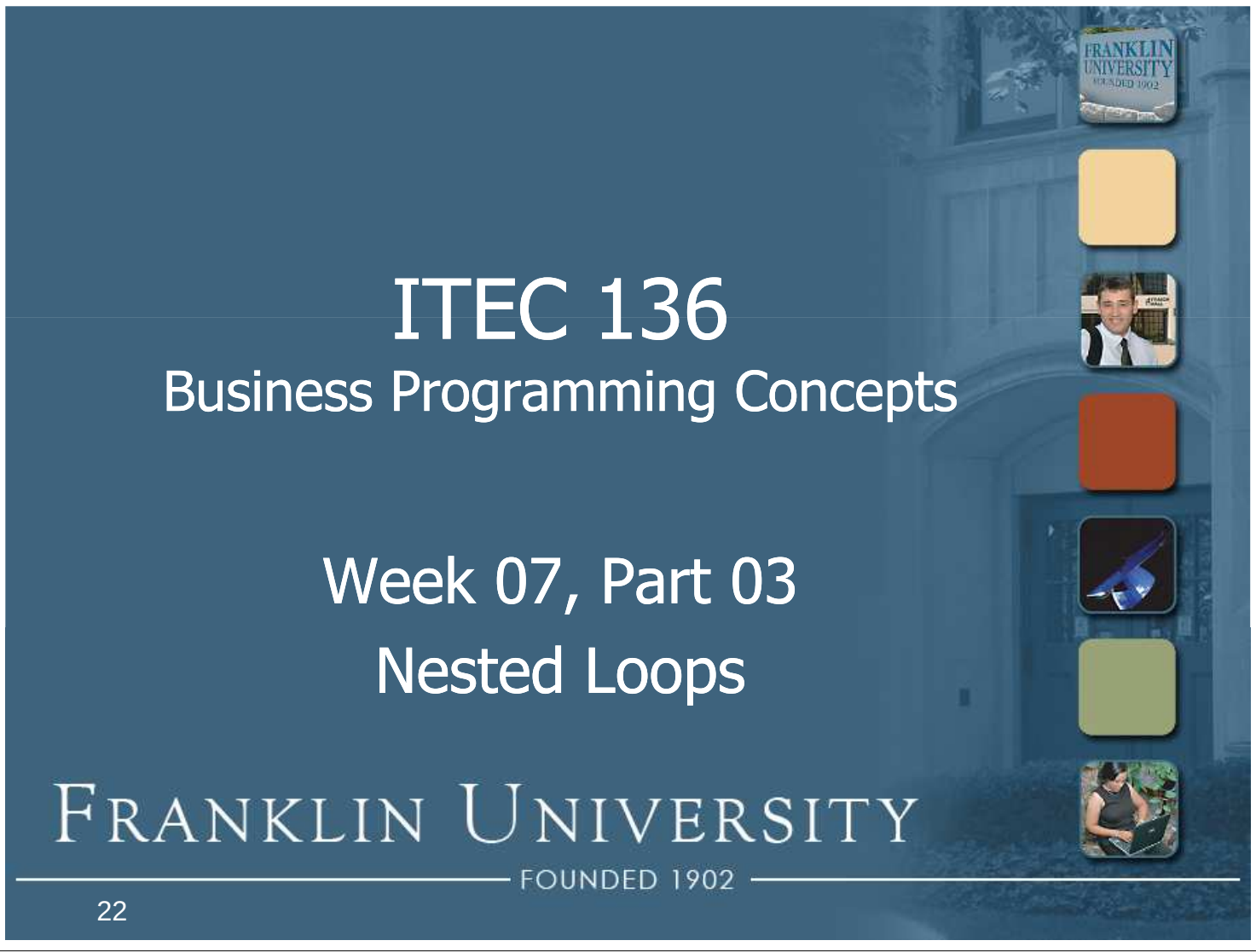
We calculate the total in the body...

Termination conditions

- Result controlled loops

```
function yearsToReach(target, principle, rate) {
    var years = 0;
    var total = 0;
    while (total < target) {
        total += principle;
        total *= (1.0 + rate)
        ++years;
    }
    return years;
}
```

...and use the result in the condition.



Nested loops

- Nested loops are loops within loops
  - Key times used: when you're not just calculating/outputting/inputting something in a straight line, but rather when it is 2-dimensional
  - Example: triangles

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


Nested loops

- Example: triangles

```
function makeTriangle1(height, ch){
    var str = "";
    for (var row = 0; row < height; ++row) {
        for (var col = 0; col < row + 1; ++col) {
            str += ch;
        }
        str += "<br />";
    }
    return str;
}
```

Nested loops

- Example: triangles

```
function makeTriangle1(height
    var str = "";
    for (var row = 0; row < height; ++row) {
        for (var col = 0; col < row + 1; ++col) {
            str += ch;
        }
        str += "<br />";
    }
    return str;
}
```

For each row in the triangle...

For each column within the row...

Nested loops

- Example: triangles

```
function makeTriangle1(height
    var str = "";
    for (var row = 0; row < height; ++row) {
        for (var col = 0; col < row + 1; ++col) {
            str += ch;
        }
        str += "<br />";
    }
    return str;
}
```

The counter in the outer loop...

...becomes part of the condition in the inner loop.

Nested loops

- Example: triangles

```
function makeTriangle1(height, ch){
    var str = "";
    for (var row = 0; row < height; ++row) {
        for (var col = 0; col < row + 1; ++col) {
            str += ch;
        }
        str += "<br />";
    }
    return str;
}
```

What would this look like if we substituted "height" for "row + 1"

Nested loops

- Hiding nested loops: functions
  - Function A has a loop, and within that loop, it calls function B
  - Function B has a loop. Therefore this situation is a loop-within-a-loop, but it doesn't look as complicated!

Nested loops

- Hiding nested loops: functions

```
function isPrime(num) {
    if (num % 2 == 0) {
        return false;
    }
    for (var i = 3; i < Math.sqrt(num); i += 2) {
        if (num % i == 0) {
            return false;
        }
    }
    return true;
}
```


Nested loops

- Hiding nested loops: functions

```
function isPrime(num) {
    if (num % 2 == 0) {
        return false;
    }
    for (va
        if
        }
    }
    return
}
```

```
function primesBetween(start, end) {
    for (var i = start; i < end; ++i) {
        if (isPrime(i)) {
            document.writeln(i + "<br />");
        }
    }
}
```

Nested loops

- Hiding nested loops: functions

```
function isPrime(num) {
    if (num % 2 == 0) {
        return false;
    }
    for (va
        if
        }
    }
    retur
}
```

```
function primesBetween(start, end) {
    for (var i = start; i < end; ++i) {
        if (isPrime(i)) {
            document.writeln(i + "<br />");
```

The call to `isPrime` is inside a loop...

And `isPrime` has a loop. Therefore, this is a nested loop in disguise.

A semi-complicated example

- Printing out a calendar
  - Does this involve a nested loops? Why or why not?
  - Given: number of days in month, and a starting day, print the calendar.

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Nested loops

- Printing a calendar

```
function makeCalendar(days, startDay){
    var str = "<table border='1'><tr>";
    var i, j;
    for (i = 0; i < startDay - 1; ++i) {
        str += "<td> </td>"
    }
    for (j = 0; j < days; ++j, ++i) {
        if (i % 7 == 0) {
            str += "</tr><tr>"
        }
        str += "<td>" + (j + 1) + "</td>";
    }
```


Nested loops

- Printing a calendar

```
function makeCalendar(days, startDay){
    var str = "<table border='1'><tr>";
    var i, j;
    for (i = 0; i < startDay - 1; ++i) {
        str += "<td> </td>"
    }
    for (j = 0; j < days; ++j, ++i)
        if (i % 7 == 0) {
            str += "</tr><tr>"
        }
        str += "<td>" + (j + 1) + "</td>";
    }
```

Prints the leading "empty" boxes

Nested loops

- Printing a calendar

```
function makeCalendar(days, startDay){
    var str = "<table border='1'><tr>";

    for (i = 0; i < startDay - 1; ++i) {
        str += "<td> </td>"
    }
    for (j = 0; j < days; ++j, ++i) {
        if (i % 7 == 0) {
            str += "</tr><tr>"
        }
        str += "<td>" + (j + 1) + "</td>";
    }
```

Prints the "filled" boxes

Prints the leading "empty" boxes

Nested loops

- Printing a calendar

```
    while (i % 7 != 0) {
        str += "<td> </td>";
        ++i;
    }
    str += "</tr></table>"
    return str;
}
```

Nested loops

- Printing a calendar

```
    while (i % 7 != 0) {
        str += "<td> </td>"
        ++i;
    }
    str += "</tr></table>"
    return str;
}
```

Prints the trailing "empty" boxes

Nested loops

- Printing a calendar
  - Just because something is 2-D in the "real world" doesn't mean that the problem necessarily involves nested loops!

ITEC 136

Business Programming Concepts

Week 07, Part 04

Changing control flow

FRANKLIN UNIVERSITY

FOUNDED 1902

Changing control flow

- Three keywords alter the flow of control in a loop:
  - `break` – this keyword immediately stops executing the loop, and jumps out to the next statement following the loop.

FRANKLIN UNIVERSITY

Changing control flow

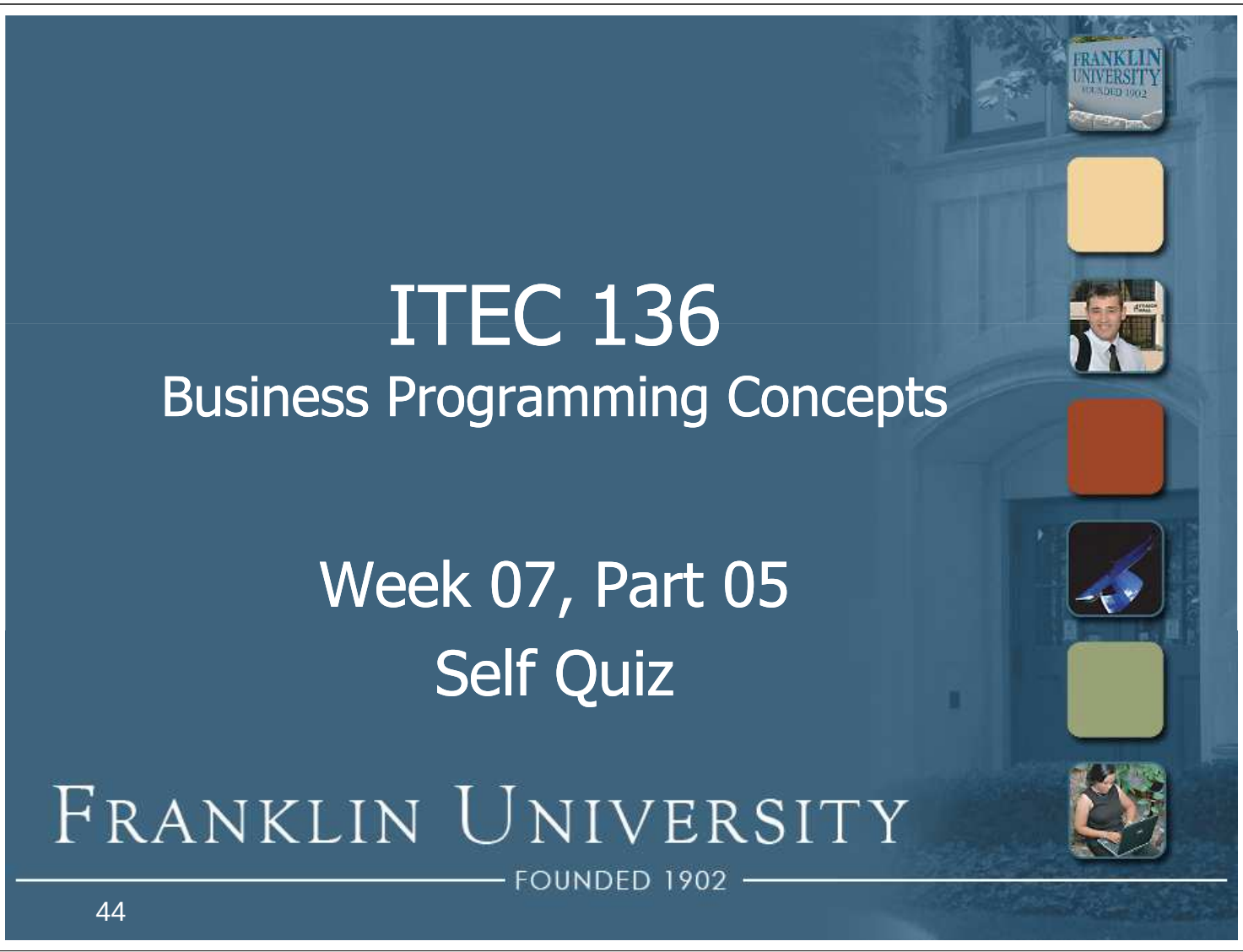
- Three keywords alter the flow of control in a loop:
  - `continue` – this keyword immediately stops executing the current iteration of the body, and cycles back to the top to test the condition again.

Changing control flow

- Three keywords alter the flow of control in a loop:
  - `return` – this keyword immediately stops executing the entire function, and returns to the next statement following the function call.

Questions?

FRANKLIN UNIVERSITY



ITEC 136

Business Programming Concepts

Week 07, Part 05

Self Quiz

FRANKLIN UNIVERSITY

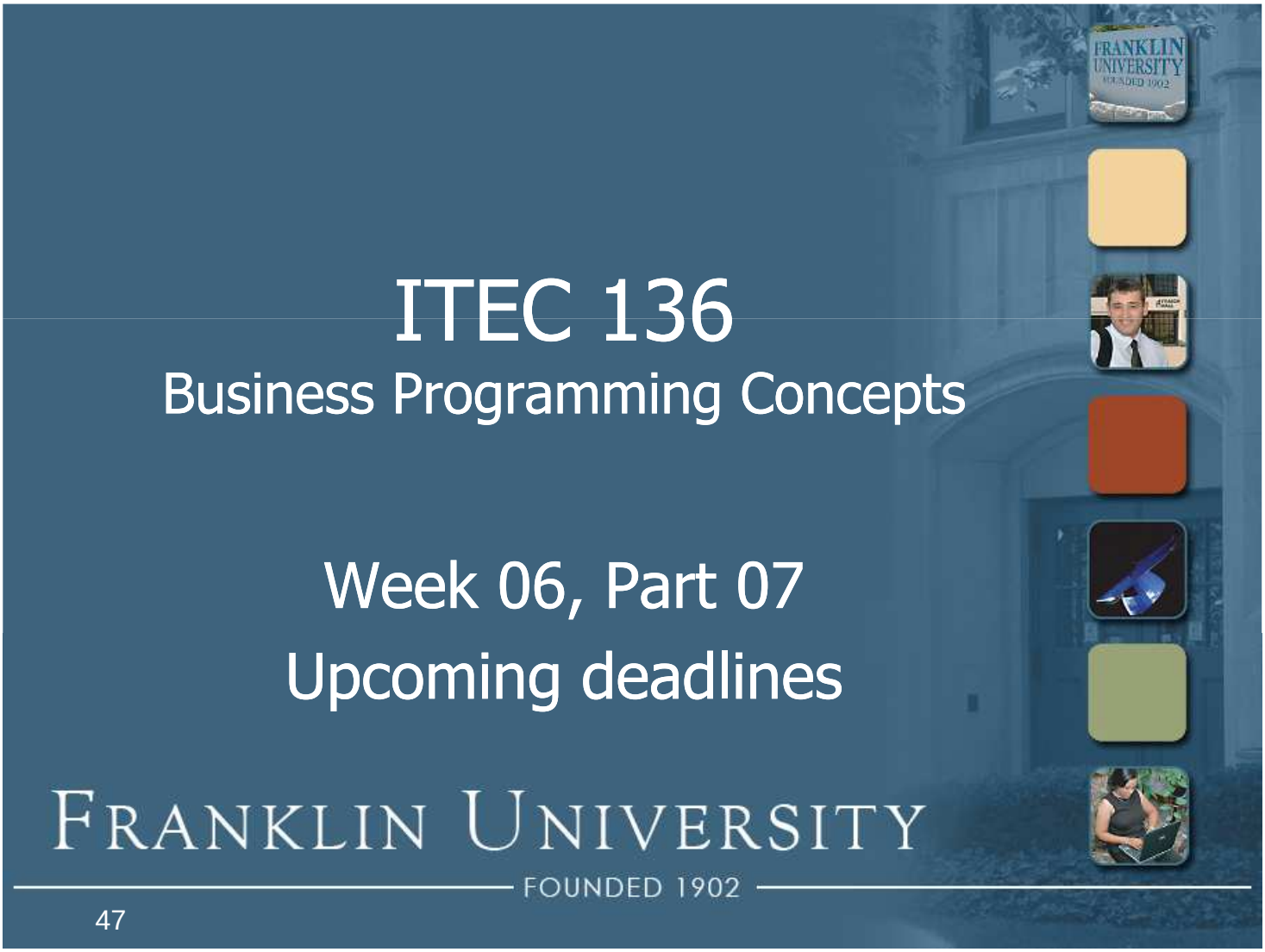
FOUNDED 1902

Self Quiz

- What is the key idea behind nested loops?
- What three keywords alter the flow of control in a loop?
- How do we hide the complexity of nested loops?

Self Quiz

- Given the `makeCalendar` function, can you write the code that will print out a yearly calendar with month names?



Upcoming Deadlines

- Homework 6 – Due February 23
- Lab 2 – Due February 23