ITEC 136
Business Programming Concepts

Week 03, Part 01
Overview

FRANKLIN UNIVERSITY
FOUNDED 1902

Week 3 Overview

- Week 2 review
  - Software Lifecycle
    - Waterfall model
    - Spiral model
  - Variables
    - Name (identifier)
    - Data type
    - Value
    - Scope

Week 3 Overview

- Week 2 review
  - Operators
    - Arithmetic
    - Relational
    - Logical

FRANKLIN UNIVERSITY

Week 3 Overview

- Outcomes
  - Describe the advantages and techniques of modularized programs.
  - Decompose a problem into modularized components.
  - Write and call functions that utilize parameters and return values.

FRANKLIN UNIVERSITY

ITEC 136

Business Programming Concepts

Week 03, Part 02

Modularized Programs

FRANKLIN UNIVERSITY

FOUNDED 1902

Algorithm

- What is an algorithm?
  - A well-defined sequence of steps that is used to solve a specific problem

Problem Solving Supplement

- Read “Problem Solving Supplement”
- Available as
  - Word document on Course web site in Module 3 Key Points 3.1
  - "ProbSolveSupplement.doc" on Course CD

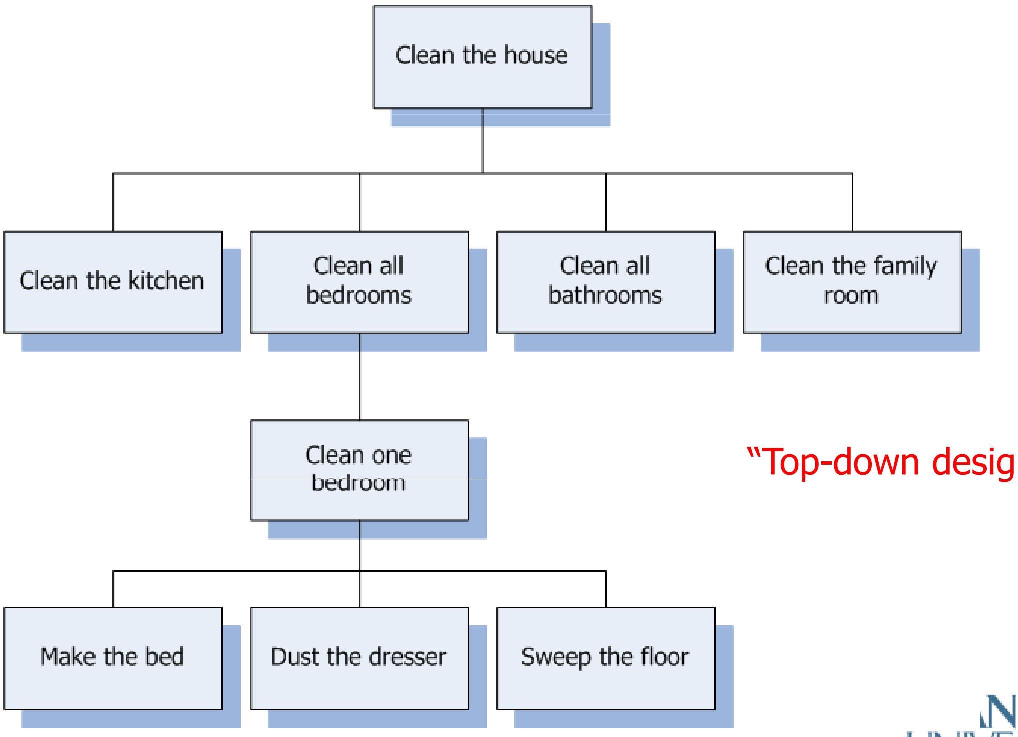
Four Step Problem Solving

1. Identify general logical chunks
2. Refine each logical chunk into more logical chunks (if possible)
3. Add detail to each logical chunk
4. Organize the chunks into the appropriate order

Modularized Programs

- Functional decomposition
  - Take big tasks and break them down into successively smaller tasks.
    - A very natural way to work
    - Ex: “Clean the house” algorithm

Modularized Programs

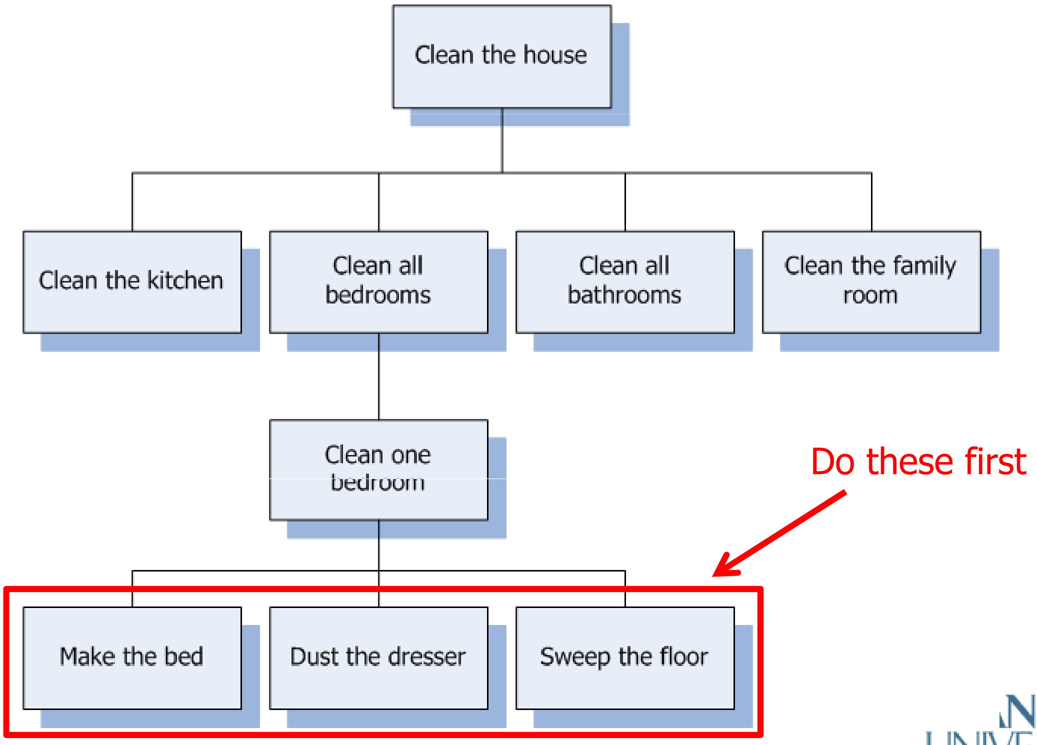


“Top-down design”

Modularized Programs

- Functional decomposition
  - Take big tasks and break them down into successively smaller tasks.
  - Perform the smaller tasks working your way back up the tree.

Modularized Programs



Clean the house

Clean the kitchen

Clean all bedrooms

Clean all bathrooms

Clean the family room

Clean one bedroom

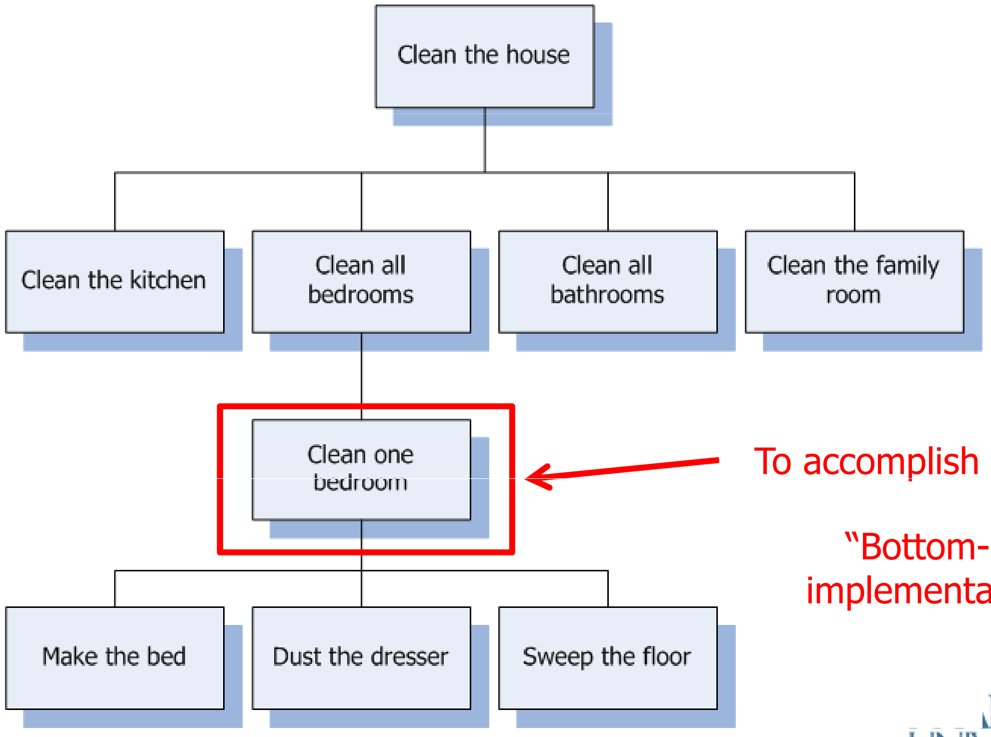
Make the bed

Dust the dresser

Sweep the floor

Do these first

Modularized Programs



Clean the house

Clean the kitchen

Clean all bedrooms

Clean all bathrooms

Clean the family room

Clean one bedroom

Make the bed

Dust the dresser

Sweep the floor

To accomplish this

“Bottom-up implementation”

A Function

- Function
  - Def: Group of related programming statements into a compact module to be called (invoked) from many other places in code
- Familiar with `writeln()` and `prompt()`
- Why? Write once, reuse many times!
- Empty function shell shown in key point 3.2 and looks like…

A Function Shell

- A shell of a function:

```
function functionName(param1, param2, ...) {
        statement#1;
        statement#2;
        ...
        statement#n;
        return someValue;
}
```

Modularized Programs

- Functional decomposition
  - Two ways to write as functions
    - Bottom-up – write the functions at the bottom level of the tree, working your way back up. Easy to test.
    - Top-down – write the “skeletons” of functions at the top level first, and “stubs” of functions at the lowest level. Easy to discern overall structure.

Modularized Programs

- Functional decomposition

```
function makeTheBed(bed) {
  // some code here that operates on bed
}

function dustTheDresser(dresser) {
  // some here that operates on dresser
}

// etc.
```

"Stubs"

Modularized Programs

- Functional decomposition

```
function cleanOneBedroom(bedroom) {
  makeTheBed(bedroom.bed);
  dustTheDresser(bedroom.dresser);
  sweepTheFloor(bedroom.floor);
}
```

"Skeleton"

Modularized Programs

- Functional decomposition

```
function cleanAllBedrooms(bedroomList) {
  foreach (bedroom in bedroomList) {
    cleanOneBedroom(bedroom);
  }
}
```

Modularized Programs

- Functional decomposition

```
function cleanTheHouse(house) {
  cleanTheKitchen(house.kitchen);
  cleanAllBedrooms(house.bedroomList);
  cleanAllBathrooms(house.bathroomList);
  cleanTheFamilyRoom(house.familyRoom);
}
```

Modularized Programs

- Advantages
  - "Working set" for developers is smaller
  - Code reuse across many modules (utility functions, etc)
  - Ease of testing
  - Clean lines of separation for teamwork

ITEC 136

Business Programming Concepts

Week 03, Part 03

Functions

FRANKLIN UNIVERSITY

FOUNDED 1902

Calling & Writing Functions

- Calling functions
  - Syntax:

```
var result = doSomething(param1, param2);
```

```
var result = someObject.doSomething(param1,
param2);
```

Calling & Writing Functions

- Writing functions
  - Syntax:

```
function doSomething(param1, param2) {
    var someResult = 0;
    // some statements;
    return someResult;
}
```

Calling & Writing Functions

- Writing functions
  - Syntax:

```
var doSomething = function(param1, param2) {
    var someResult = 0;
    // some statements;
    return someResult;
}
```

Calling & Writing Functions

- Example
  - A function that will “bold” text

```
function makeBold(text) {
    var result = "<b>" + text + "</b>";
    return result;
}
```

Calling & Writing Functions

- Function Composition
  - Using the return value from one function as a parameter to another

```
document.writeln(makeBold("Hello World!"));
```

Calling & Writing Functions

- Example
  - Function to return average of 3 numbers:

Calling & Writing Functions

- Example
  - Function to return the maximum of 3 numbers (hint: use the “?:” operator):

Calling & Writing Functions

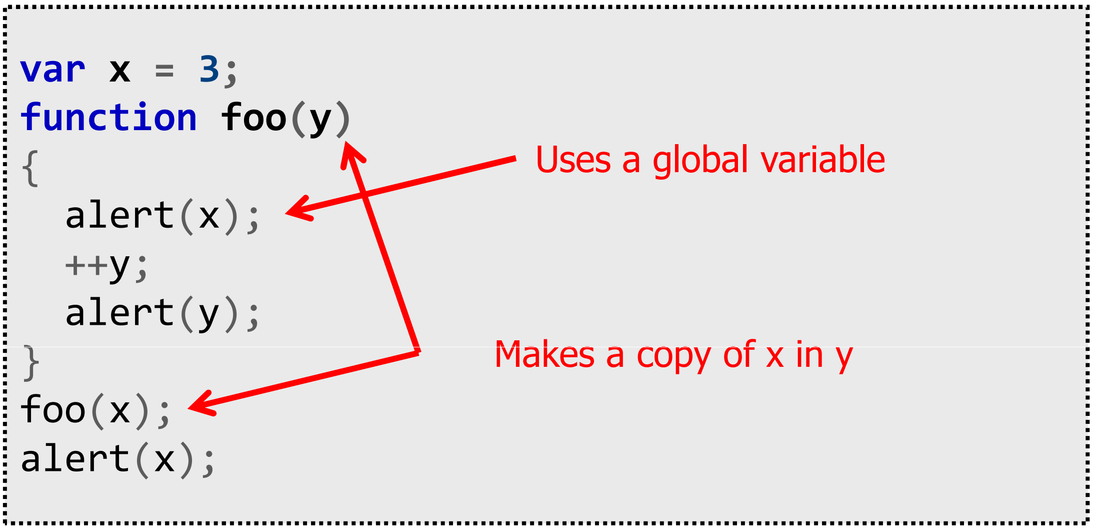
- Example
  - Function to convert a Fahrenheit parameter into Celsius

Calling & Writing Functions

- Variable scope
  - “Scope” is a range of lines during which the variable is able to be used.
  - A variable declared using “**`var`**” within a function is inaccessible from outside the function. Called “local variables”
  - Parameters are just like local variables
  - Global variables == BAD!

Calling & Writing Functions

- Parameters are passed by value



```
var x = 3;
function foo(y)
{
   alert(x);
   ++y;
   alert(y);
}
foo(x);
alert(x);
```

Uses a global variable

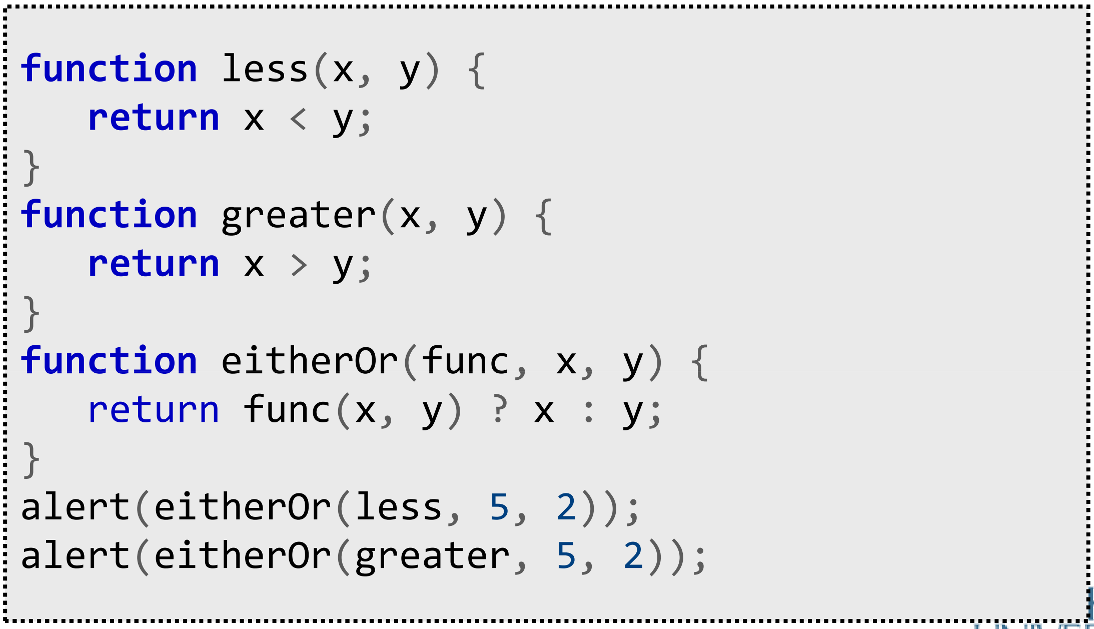
Makes a copy of x in y

Calling & Writing Functions

- Functions as parameters
  - Functions are themselves variables.
  - Any variable can be passed as a parameter to a function.
  - Therefore, a function can be passed to another function

Calling & Writing Functions

- Functions as parameters



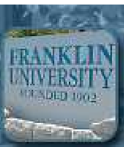
```
function less(x, y) {
   return x < y;
}
function greater(x, y) {
   return x > y;
}
function eitherOr(func, x, y) {
   return func(x, y) ? x : y;
}
alert(eitherOr(less, 5, 2));
alert(eitherOr(greater, 5, 2));
```

Calling & Writing Functions

- Functions as return values

```
function countUpFrom(x) {
  var y = x;
  return function() {
    alert(y);
    ++y;
  }
}
var myFunction = countUpFrom(8);
myFunction();
myFunction();
```

A “closure”



ITEC 136

Business Programming Concepts

Week 03, Part 04

Event Handlers

FRANKLIN UNIVERSITY

FOUNDED 1902

Event Handlers

- Events
  - Generated in response to user actions
    - Button clicks
    - Mouse overs
    - Focus/blur
    - Keypresses
    - And many others

Event Handlers

- Events
  - Generally want something to happen when the user generates an event.
  - Use the `<input>` tag to create UI elements and the "`onXXX()`" attributes to associate an event handler.

Event Handlers

- Events
  - Example:

```
<input type="button" value="Click me!"
onclick="alert('Nice click.')" />
```

Event Handlers

- Events
  - Generally, `<input>` tags are found within a `<form>` tag, but not exclusively.
  - "`type`" attribute of `<input>` defines what kind of UI control is displayed
    - button, text, textarea, select, etc.

Event Handlers

- Accessing UI elements
  - Be sure to assign the “`id`” attribute to all `<input>` elements.
  - Use `document.getElementById()` to get access to the UI element.
  - Read from or assign something to the element’s “`value`” property.

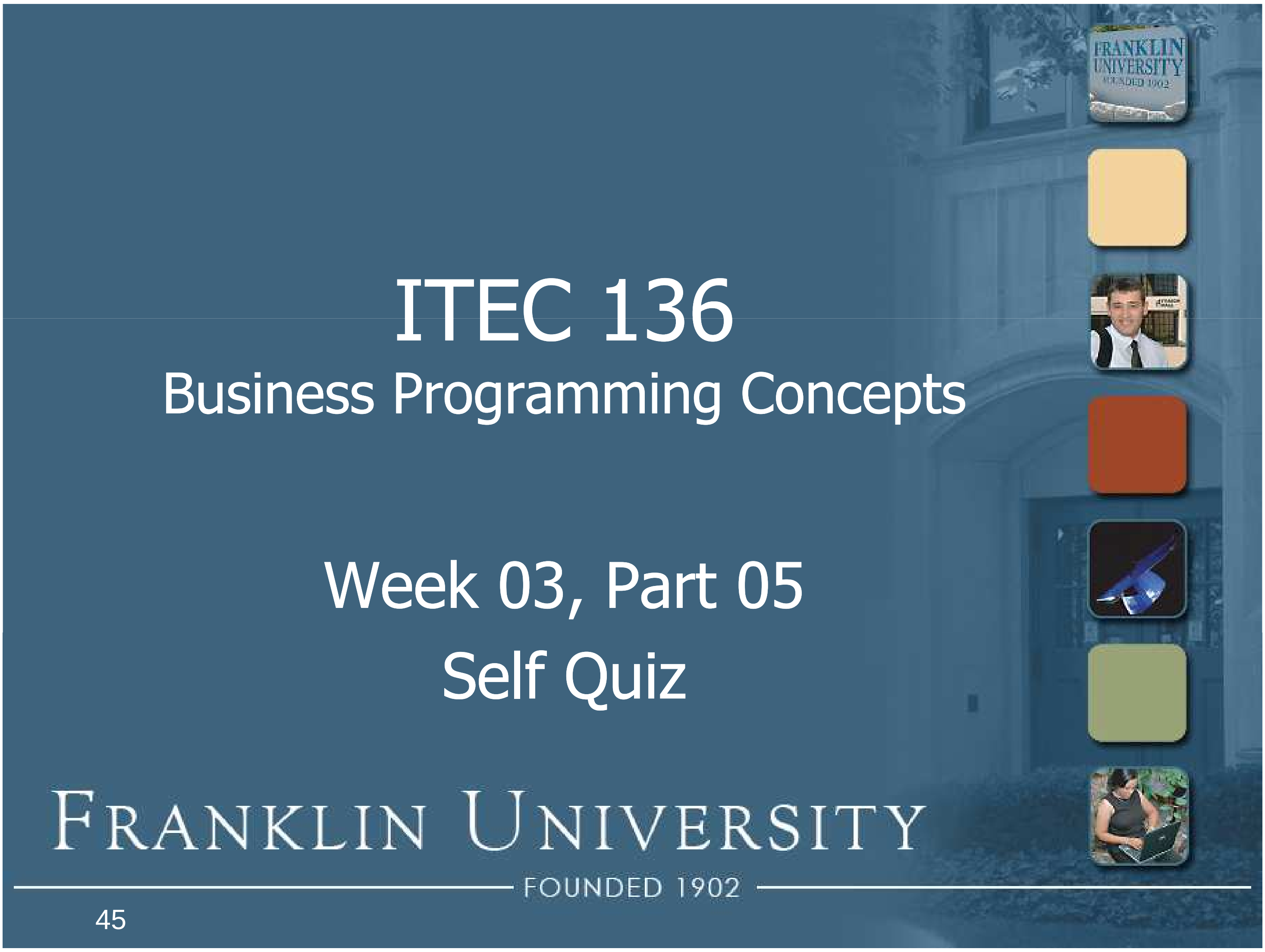
Event Handlers

- Events
  - Example: incrementing counter
    http://cs.franklin.edu/~whittakt/ITEC136/examples/Counter.html

Event Handlers

- Events
  - Try converting counter into a PIN entry pad
  - Try writing a Fahrenheit to Celsius conversion using event-driven programming with functions.

Questions?



ITEC 136

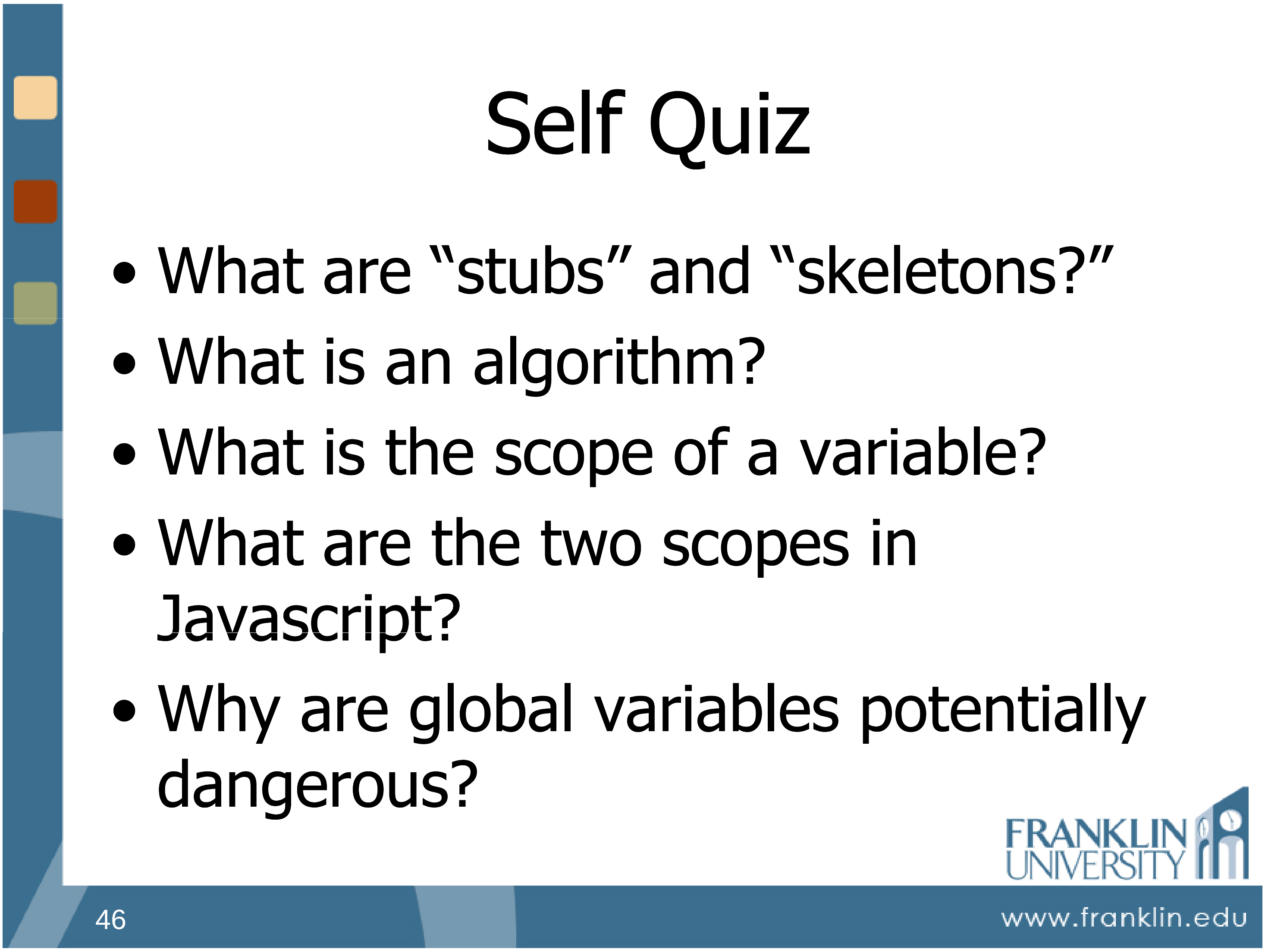
Business Programming Concepts

Week 03, Part 05

Self Quiz

FRANKLIN UNIVERSITY

FOUNDED 1902



Self Quiz

- What are “stubs” and “skeletons?”
- What is an algorithm?
- What is the scope of a variable?
- What are the two scopes in Javascript?
- Why are global variables potentially dangerous?

FRANKLIN UNIVERSITY

Self Quiz

- Why do we write code inside functions?
- Write a function that computes the body mass index of a person using the height and weight as parameters.

ITEC 136

Business Programming Concepts

Week 03, Part 06

Upcoming deadlines

FRANKLIN UNIVERSITY

FOUNDED 1902

Upcoming Deadlines

- Homework 3 – Due January 26
- Pre-class 4 – Due January 26
- Lab 1 – Due February 2
- Exam 1 – In class February 2
- Reflection paper draft 1 – Due February 2