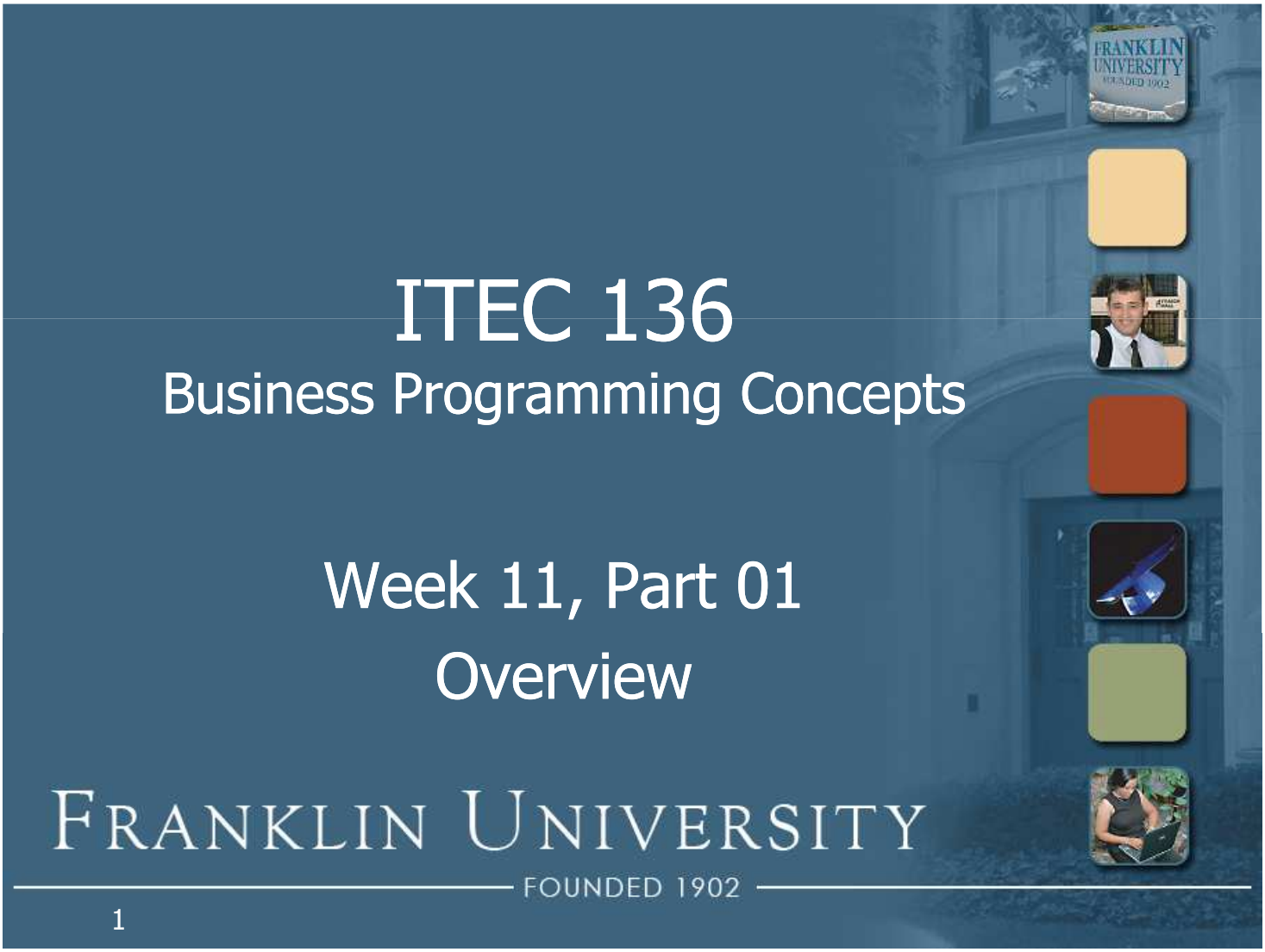


Week 11 Overview

- Week 9 review
  - Object references
  - Built-in objects
    - Date
    - String
    - Number
    - Math

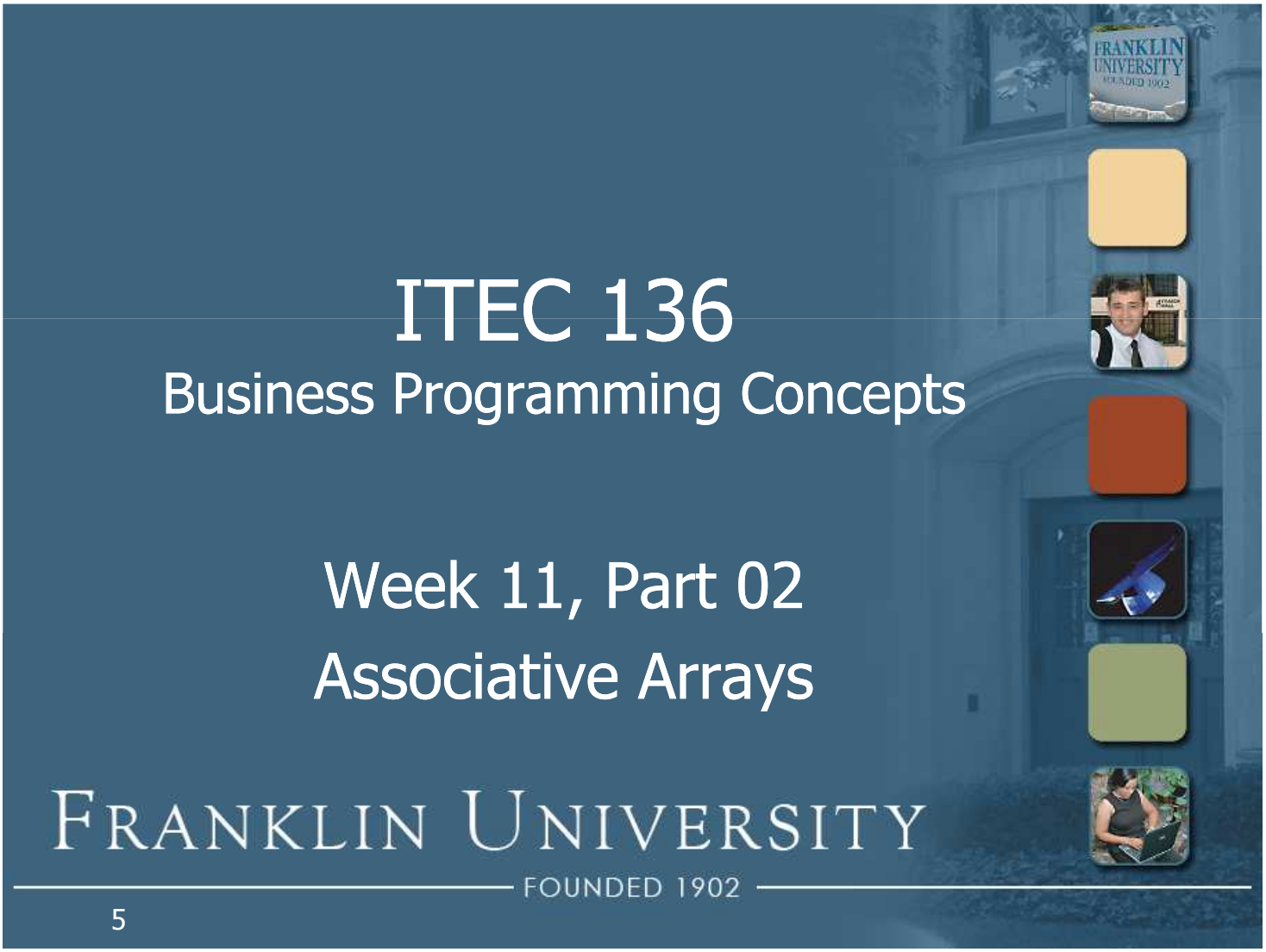
See the documentation for details!

Week 11 Overview

- Week 9 review
  - Arrays
    - `length` property
    - `for…in` loops
    - `in` operator tests array index membership
    - Arrays grow as needed
    - Arrays can be sparse
    - Many methods – see the documentation!

Week 11 Overview

- Outcomes
  - Insert, remove, and search array-based data.
  - Compare and contrast linear and binary search algorithms.
  - Work with associative arrays.



Associative Arrays

- Associative vs. Standard Arrays
  - Standard arrays
    - Indexed using an integer [0, *n* - 1] for array of length *n*.
    - Find items in the array fast when you already know the index. Otherwise, you need to search.
    - What about alphabetic lookups (e.g. a phone directory)?

Associative Arrays

- Associative vs. Standard Arrays
  - Associative arrays
    - Use any arbitrary object as an index value (strings, most commonly).
    - Don't use the `Array` constructor, but rather the `Object` constructor instead.

Associative Arrays

- Ex: Using associative arrays

```
var obj = new Object();
obj["name"] = "John Smith";
obj["birthday"] = new Date(1981, 7, 16);
obj["gpa"] = 3.84;
obj.major = "ITEC";
var alertStr = "";
for (property in obj) {
  alertStr += property + ":" + obj[property] + "\n";
}
alert(alertStr);
```

Associative Arrays

- Ex: Using associative arrays

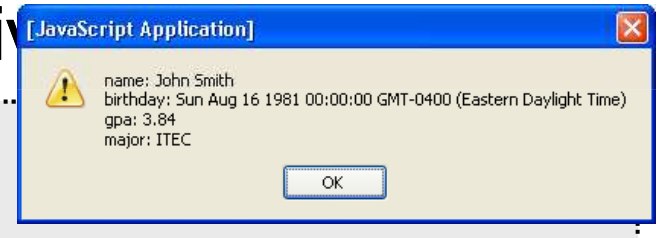
```
var obj = new Object();
obj["name"] = "John Smith";
obj["birthday"] = new Date(1981, 7, 16);
obj["gpa"] = 3.84;
obj.major = "ITEC";
var alertStr = "";
for (property in obj) {
  alertStr += property + ":" + obj[property] + "\n";
}
alert(alertStr);
```

Notice, use `Object()` constructor, not `Array()` constructor.

Can use both the subscript notation with strings or the dot notation.

Associative Arrays

- Ex: Using associati




```
var obj = new Object();
obj["name"] = "John Smith";
obj["birthday"] = new Date(1981, 7, 16);
obj["gpa"] = 3.84;
obj.major = "ITEC";
var alertStr = "";
for (property in obj) {
  alertStr += property + ":" + obj[property] + "\n";
}
alert(alertStr);
```

Associative Arrays

- Shortcuts:

```
// two ways to create an empty Object
var obj1 = new Object();
var obj2 = { };
```

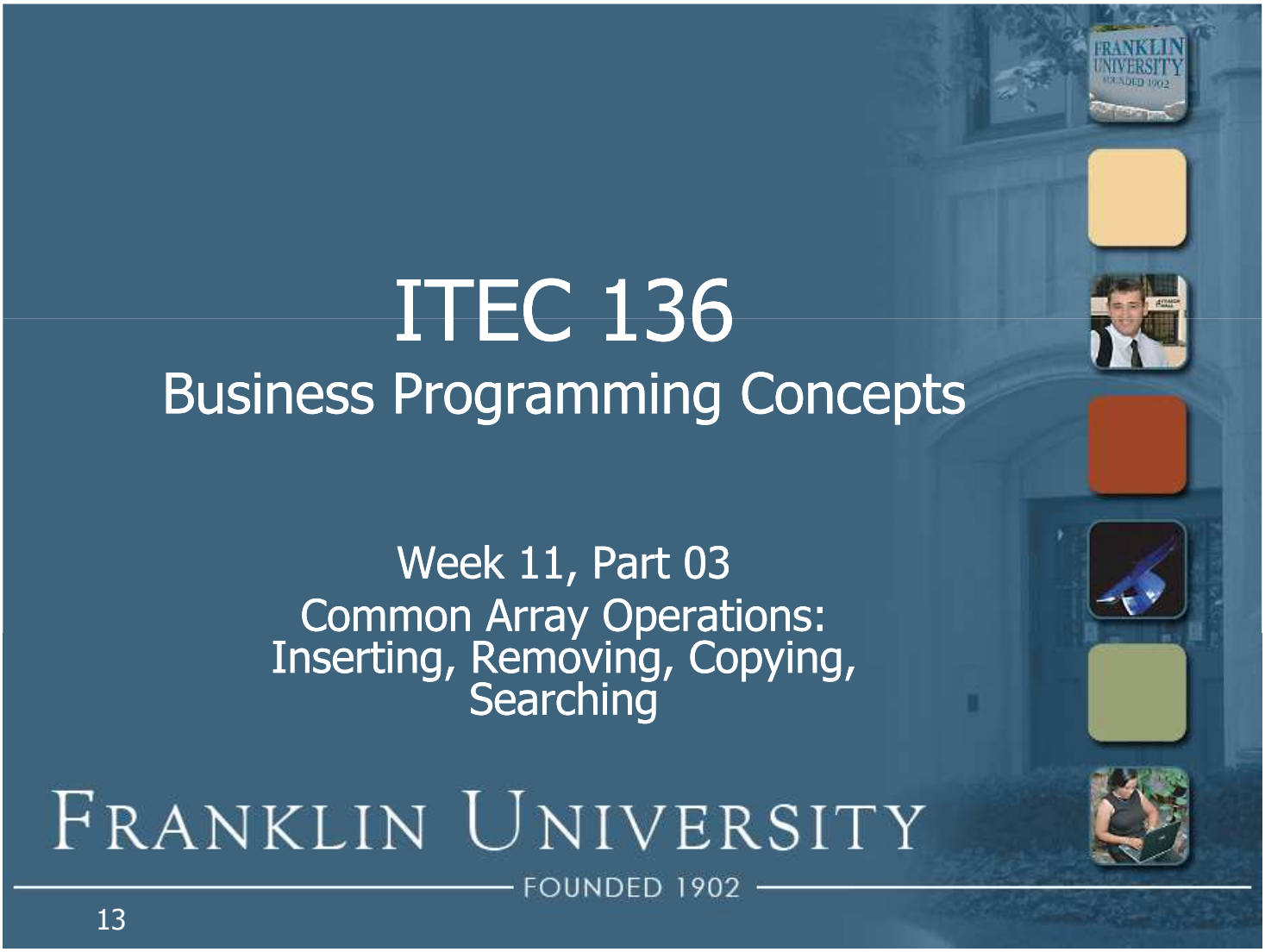
```
// two ways to create Object properties
var obj = new Object();
obj.prop1 = 42;
obj["prop2"] = "Life, the Universe, and Everything";
```


Associative Arrays

- Shortcuts:

```
// Initializing objects
var obj = {
    "prop1" : 42,
     prop2  : "Life, the Universe and Everything"
}
```

Used as a property name regardless of quotes. Not interpreted as a variable.



Common Array Operations

- Collections of data
  - **Add**: put a new element into the collection
  - **Remove**: take an element out of the collection
  - **Search**: determine if (or where) an element exists in the collection
  - **Sort**: order elements according to some criterion – next week.

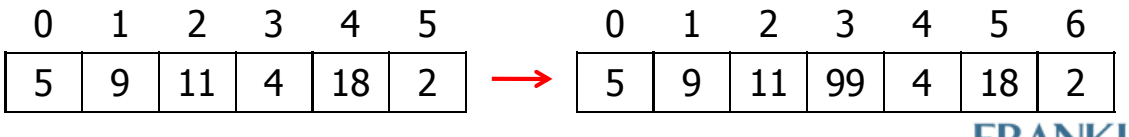
Common Array Operations

- Adding to the end of an array
  - JavaScript arrays grow to accommodate new elements.
  - Assign the value into the array at the index equivalent to the length. E.g.

```
// appending to an array
arr[arr.length] = someNewValue;
// or...
arr.push(someNewValue);
```


Common Array Operations

- Adding at an arbitrary array index
  - Must "slide" each element to the right by one to open up space for the new value. Stop sliding when a free slot is found (usually at the end of the array).
  - Ex: add 99 at index 3:



Common Array Operations

- Adding at an arbitrary array index

```
function insertAtIndex(arr, element, index) {
    do {
        var temp = arr[index];
        arr[index] = element;
        element = temp;
        ++index;
    } while (element != undefined);
}
```


Common Array Operations

- Removing at an arbitrary index
  - Reverse the previous operation. Slide elements to the left.

Common Array Operations

- Removing at an arbitrary index

```
function removeAtIndex(arr, index) {
    var element = arr[index];
    if (index in arr) {
        while (index + 1 < arr.length) {
            arr[index] = arr[index + 1];
            ++index;
        }
        arr.pop();
    }
    return element;
}
```


Common Array Operations

- Removing at an arbitrary index

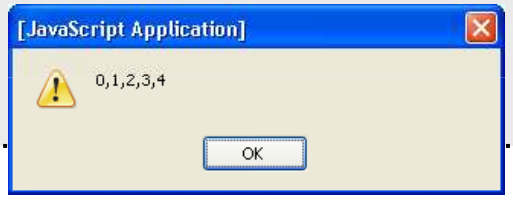
```
function removeAtIndex(arr, index) {
    var element = arr[index];
    if (index in arr) {
        while (index + 1 < arr.length) {
            arr[index] = arr[index + 1];
            ++index;
        }
        arr.pop();
    }
    return element;
}
```

Removes the last element of the array (a duplicate).

Common Array Operations

- Copying an array
  - Recall assignment of objects results in a *shallow copy* of the reference

```
var arr1 = [0, 1, 2, 3, 4, 5];
var arr2 = arr1;
arr1.pop();
alert(arr2);
```



Common Array Operations

- Copying an array
  - Recall assignment of objects results in a *shallow copy* of the reference
  - To create a *deep copy* involves creating an entirely new array, and copying over all the elements.

Common Array Operations

- Copying an array

```
// first attempt -- one level deep
function arrayCopy(arr) {
   var result = new Array();
   for (index in arr) {
      result[index] = arr[index];
   }
   return result;
}
```


Common Array Operations

- Copying an array

```
// first attempt -- one level deep
function arrayCopy(arr) {
   var result = new Array();
   for (index in arr) {
      result[index] = arr[index];
   }
   return result;
}
```

Problem: what if this element is itself an array (an array within an array)?

Common Array Operations

- Copying an array

```
// recursive deep copy
function arrayCopy(arr)
{
   var result = new Array();
   for (index in arr) {
      if (arr[index] instanceof Array)
         result[index] = arrayCopy(arr[index]);
      else
         result[index] = arr[index];
   }
   return result;
}
```

`instanceof`: an operator that checks to see if the left-hand operand is of a compatible type with the right-hand operand.

Online Examples

- More array examples online
- Many Array function examples

http://www.java2s.com/Code/JavaScript/Language-Basics/Array.htm

http://www.java2s.com/Code/JavaScript/Language-Basics/Array.htm

Common Array Operations

- Searching an array
  - Find and return the index where the element exists in the array. If the element doesn't exist in the array, return an invalid index (usually -1).

Common Array Operations

- Searching an array
  - Brute-force approach: linear search
    - Start at index zero, comparing one element against another until the match is found.
    - If no match is found, return -1 upon reaching the end of the array.

Common Array Operations

- Searching an array – linear search

```
// search an array for a matching value
function search(array, value) {
   for (var i in array) {
      if (array[i] == value) {
         return i;
      }
   }
   return -1;
}
```


Common Array Operations

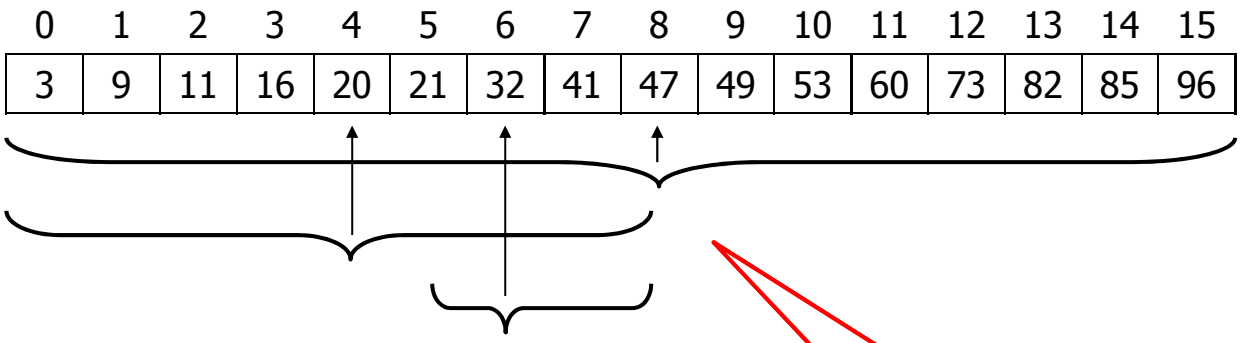
- Searching an array
  - The search can be much faster if the array is already sorted – binary search.
  - Like the High/Low game on "The Price is Right."

Common Array Operations

- Searching an array – binary search
  - Search between [left, right) bounds.
    - Pick the middle index: (left+right)/2; see if it is too high or too low.
    - If too low, search the right half of the array (i.e. [mid + 1, right))
    - Otherwise, search the right half of the array(i.e. [left, mid))

Common Array Operations

- Searching an array – binary search



Determine if 32 is in this sorted array

Common Array Operations

- Searching an array – binary search

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	9	11	16	20	21	32	41	47	49	53	60	73	82	85	96

left	right	mid	arr[mid]
0	16		

Common Array Operations

- Searching an array – binary search

```
function binarySearch(array, value, left, right) {
  while (left < right) {
    var mid = Math.floor((left + right) / 2);
    if (array[mid] < value)
      left = mid + 1;
    else if (array[mid] > value)
      right = mid;
    else
      return mid;
  }
  return -(left + 1);
}
```

Common Array Operations

- Searching an array – binary search

```
function binarySearch(array, value, left, right) {
  while (left < right) {
    var mid = Math.floor((left + right) / 2);
    if (array[mid] < value)
      left = mid + 1;
    else if (array[mid] > value)
      right = mid;
    else
      return mid;
  }
  return -(left + 1);
}
```

Right bound is not included in the search range.

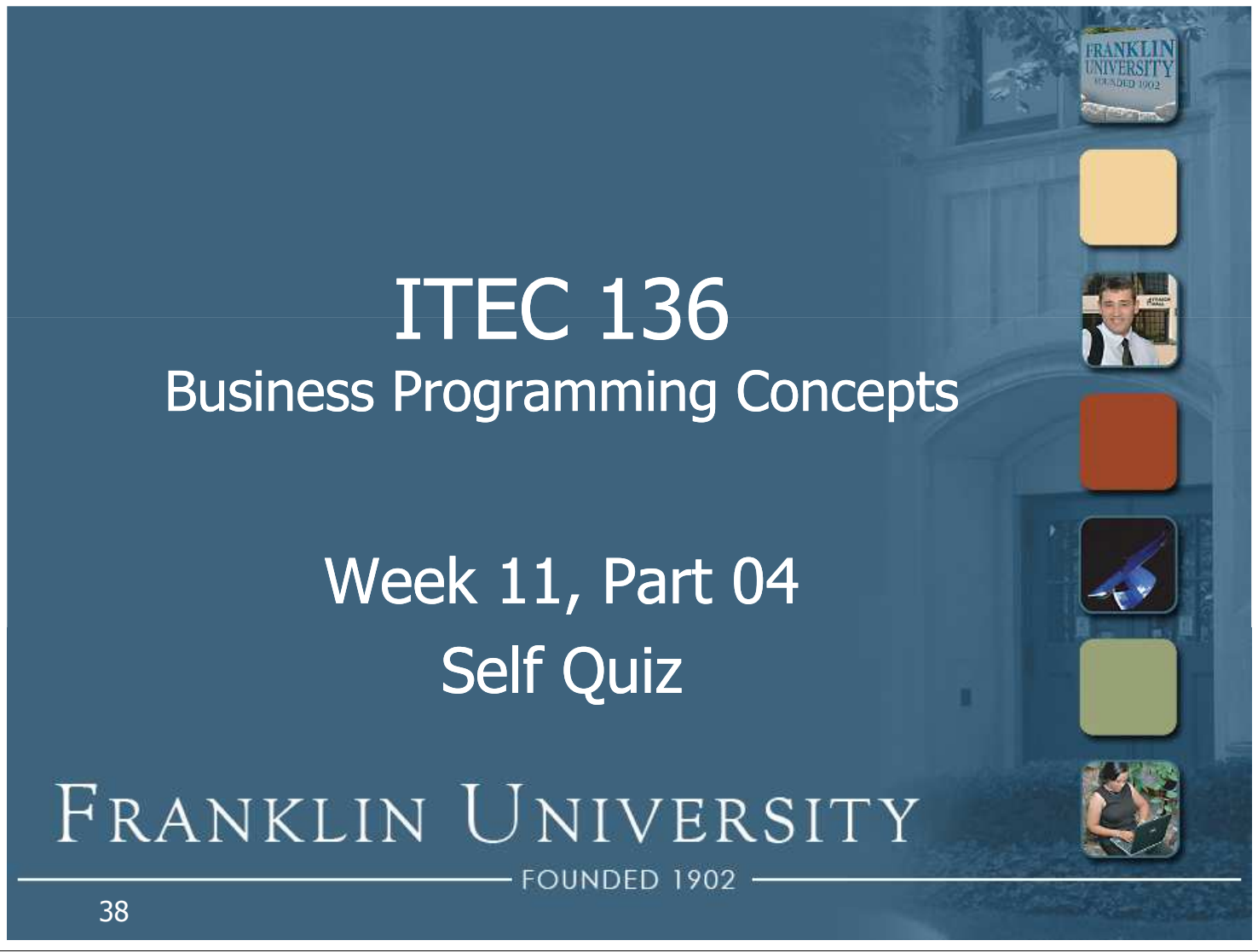
If the value had been in the array, it would have been at left+1.

Questions?

Next Week

- More arrays!
  - Sorting arrays using selection sort, insertion sort, and bubble sort.
  - Multi-dimensional arrays.

FRANKLIN UNIVERSITY



ITEC 136

Business Programming Concepts

Week 11, Part 04

Self Quiz

FRANKLIN UNIVERSITY

FOUNDED 1902

Self Quiz

- Create a function that receives a name, salary, and date of birth and returns an object with those three properties set.
- Use the enhanced for-loop to write a function `toString` that takes any object as a parameter and returns a string with all properties displayed.

Self Quiz

- Write a function `nextIndexOf` that takes an array, a starting index, and a value to search for. Starting at the given index, find the next element.

Self Quiz

- Write a function `allIndicesOf` that takes an array and a value as parameters. It should return an array containing all the indices in the parameter array that match the value.

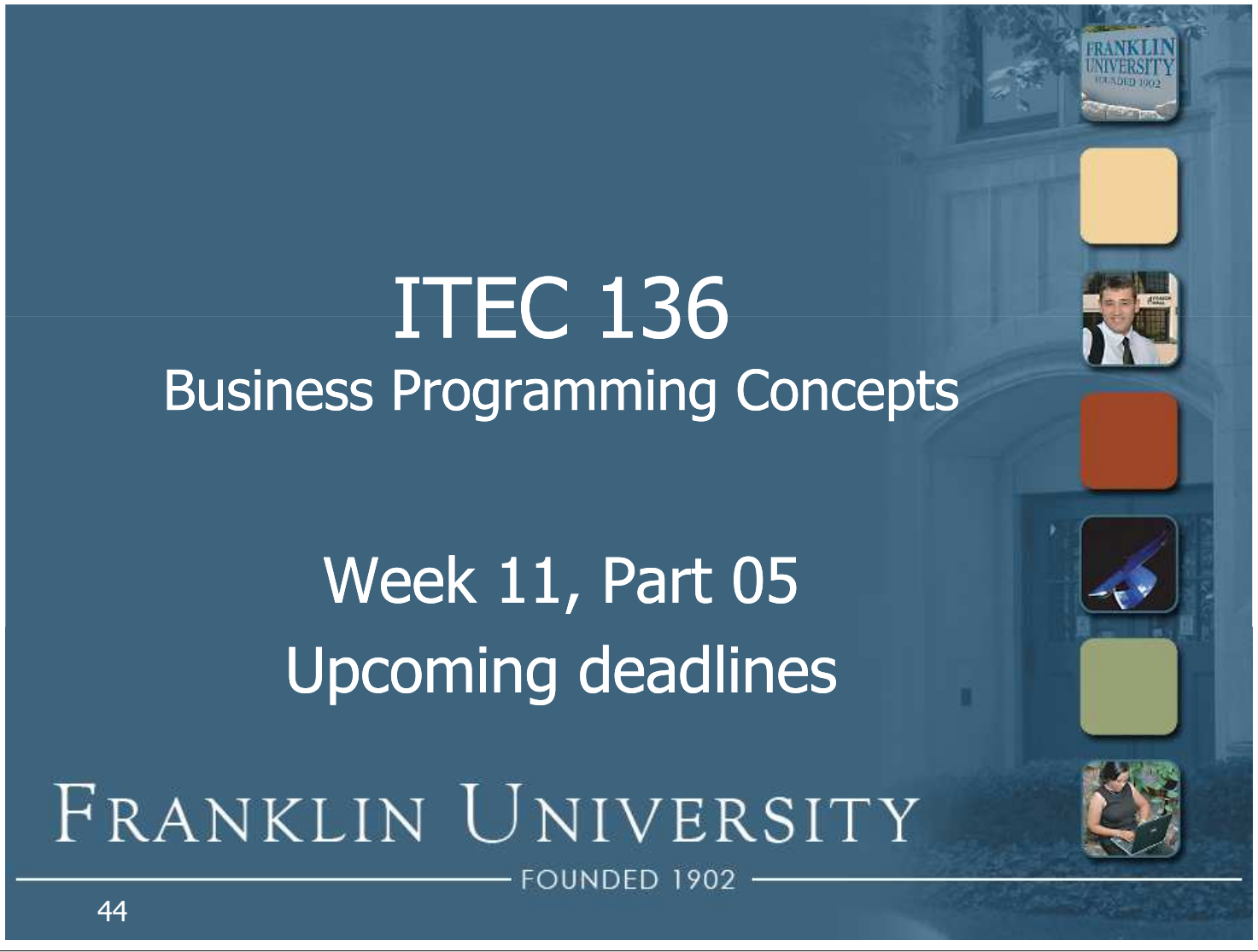
Self Quiz

- Write a function `makeHistogram` that takes an array of integers in the range [0-100] as a parameter. It should return a string representing a histogram of the data in tenths, one asterisk for each value in the array.

Self Quiz

- Write a function `letterFrequency` that takes a string as a parameter and returns an associative array containing the frequencies of each letter in the string.

FRANKLIN UNIVERSITY



ITEC 136

Business Programming Concepts

Week 11, Part 05

Upcoming deadlines

FRANKLIN UNIVERSITY

FOUNDED 1902

Upcoming Deadlines

- Pre-class 12: due March 23
- Homework 9: due March 23

FRANKLIN UNIVERSITY