CSCI-GA.3033.003
Scripting Languages

9/25/2013
Prelim 1 Review
Outline

• Associativity and Precedence
• Typing
• Properties
• Callbacks
### Concepts

#### Operator Characterization

<table>
<thead>
<tr>
<th>Operator</th>
<th>Arity</th>
<th>Associativity</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ... )</td>
<td>1 = unary</td>
<td>L = left</td>
<td>from high to low</td>
</tr>
<tr>
<td>^</td>
<td>2 = binary</td>
<td>R = right</td>
<td></td>
</tr>
<tr>
<td>+, -</td>
<td>2</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>* /</td>
<td>2</td>
<td>L (Multiplicative)</td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>2</td>
<td>L (Integer division)</td>
<td></td>
</tr>
<tr>
<td>Mod</td>
<td>2</td>
<td>L (Modulus)</td>
<td></td>
</tr>
<tr>
<td>+, -</td>
<td>2</td>
<td>L (Additive)</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>2</td>
<td>L (String concatenation)</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>2</td>
<td>L (Bit shift)</td>
<td></td>
</tr>
<tr>
<td>=, &lt;=, &lt;, &gt;, &gt;=, Is</td>
<td>2</td>
<td>L (Comparison)</td>
<td></td>
</tr>
<tr>
<td>Not</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>And, Or, Xor, Eqv, Imp</td>
<td>2</td>
<td>L (Logic (not all same precedence))</td>
<td></td>
</tr>
<tr>
<td>[Set] ... = ...</td>
<td>2</td>
<td>Assignment statement</td>
<td></td>
</tr>
</tbody>
</table>
## Concepts

### Arity, Precedence, Associativity

<table>
<thead>
<tr>
<th>Arity</th>
<th>Number of operands</th>
<th>−2</th>
<th>2 − 2</th>
<th>unary binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precedence</td>
<td>Binding strength</td>
<td>2+2*2</td>
<td>(2+2)*2</td>
<td>* has higher precedence than +</td>
</tr>
<tr>
<td>Associativity</td>
<td>Grouping direction</td>
<td>2/2/2</td>
<td>(2/2)/2</td>
<td>/ is left-associative</td>
</tr>
</tbody>
</table>

• Precedence and associativity in programming usually follows the conventions from math.
Outline

• Associativity and Precedence
• Typing
• Properties
• Callbacks
Typing

- **Strong typing** = no implicit type conversion
- **Weak typing** = implicit type conversion
- **Static typing** = check for type errors at *compile* time
- **Dynamic typing** = check for type errors at *run time*
- **Gradual typing** = checks for some errors at compiler time, some at run time. Directed by which parts of the program have explicit types. (Ex. option explicit in VBA)
Explicit typing = declare the type in your code
  – Java
Implicit typing = compiler infers the type
  – ML language,
  – VBA type declaration characters $ means string
Weak/Strong, Static/Dynamic Typing

- Strong typing (explicit conversions)
  - ML
  - Scheme
- Weak typing (implicit conversions)
  - assembler
- Static typing (compile-time checks)
  - C
  - Java
- Dynamic typing (runtime checks)
  - VBA
  - PHP
  - JavaScript
  - Perl

ML
Java
VBA
Scheme
PHP
JavaScript
Perl

CS 5142 Cornell University
9/18/13
Outline

• Associativity and Precedence
• Typing
• Properties
• Callbacks
Properties

• Read and written like fields (dot syntax)
• Accesses are translated to set/get methods
• Have easy-to-read syntax, but can implement complex functionality
• Can be indexed. Seems like an array, but associates a behavior with each read/write
Properties

Public Function GetLength() As Double
    GetLength = Sqr(X ^ 2 + Y ^ 2)
End Function

Public Sub SetLength(NewLen As Double)
    OldLen = GetLength
    If OldLen = 0 Then
        X = NewLen
        Y = 0
    Else
        X = X * NewLen / OldLen
        Y = Y * NewLen / OldLen
    End If
End Sub
Properties vs. Fields

- Both: dot notation look&feel
  - Writable: `a1.color = "red"
  - Readable: `Debug.print a1.color`
- Properties only: active (associated behavior)
  - E.g., update graphical representation
- Properties only: may be indexed, like arrays
  - `cake.ingredient("topping") = a1`
- Other languages with properties:
  - E.g., PHP, Delphi, C#
Common Uses of Properties

Simple (field-like)
- Visual update
- Invariant checking
  - Filter illegal values
  - Read-only
  - Copy on write
- Logging

Indexed (array-like)
- Collections
  - Resizable array
  - Hash map
- Persistence
  - File
  - Database
  - Cookie
Collections

- Dim col As Slides
- Set col = ActivePresentation.Slides
- Dim i As Integer
- Debug.Print "for-loop, indexed property access"
- For i = 1 To col.Count
  - Debug.Print col.Item(i).Name
- Next i
- Debug.Print "for-loop, default property access"
- For i = 1 To col.Count
  - Debug.Print col(i).Name
- Next i
- Dim s As Slide
- Debug.Print "for-each loop"
- For Each s In col
  - Debug.Print s.Name
- Next s
Powerpoint Object Model

• The complete object model is much larger
• See Visual Basic help in editor
• Also in MSDN library:
  → Office development
  → Microsoft Office 2003
  → Office 2003
  → VBA reference
  → Powerpoint help
  → Object model
Object Model

- Object-oriented API for embedded scripts
- Other examples:
  - Object models for other Microsoft apps
  - DOM = document object model for XML

• Concepts

• Application
  - CommandBars
  - CommandBar
  - DocumentWindows
  - Selection
  - ShapeRange >
  - Presentations
    - Presentation >
    - ActiveWindow
      - Selection
      - SlideRange
      - Shapes
        - Shape
        - ActivePresentation
          - Slides
          - Slide

• Gray box: simple object
• White box: collection

• Top-level “creatable objects”
• “has-a” / “contains” relationship
• “>”: more objects (not shown)
Object Model Usage

Dim S As PowerPoint.Slide
Set S = ActivePresentation.Slides( _
   ActivePresentation.Slides.Count)
Outline

• Associativity and Precedence
• Typing
• Properties
• Callbacks
Callbacks

• A *callback* is a function or block of code that is passed to some other code as a parameter.

• It is expected that the callee will “call back” to the called at the appropriate time.
## Callback Mechanisms

<table>
<thead>
<tr>
<th>Environment</th>
<th>Method of Passing Callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBA form</td>
<td>Subroutine in form with mangled name</td>
</tr>
<tr>
<td>VBA class</td>
<td>WithEvent / RaiseEvent statements</td>
</tr>
<tr>
<td>Java</td>
<td>Pass object on which to call method</td>
</tr>
<tr>
<td>Perl, Python, JavaScript</td>
<td>Pass anonymous function (lambda)</td>
</tr>
<tr>
<td>C, C++</td>
<td>Pass function pointer</td>
</tr>
<tr>
<td>C++</td>
<td>Pass object on which to call “()” operator</td>
</tr>
<tr>
<td>SmallTalk</td>
<td>Pass code block</td>
</tr>
<tr>
<td>PHP</td>
<td>Pass name of function as string</td>
</tr>
</tbody>
</table>
• Concepts

Call-backs

- VBA code
  - frmLemonStar.Show()
  - cmdPaint_Click()
  - return from handler
  - return from “Show”

- Interpreter
  - wait for user input
  - edit text box
  - click button

- User
Last Slide

• Good luck!
• Bring a pencil or pen
• Closed book. No notes, phones, etc.