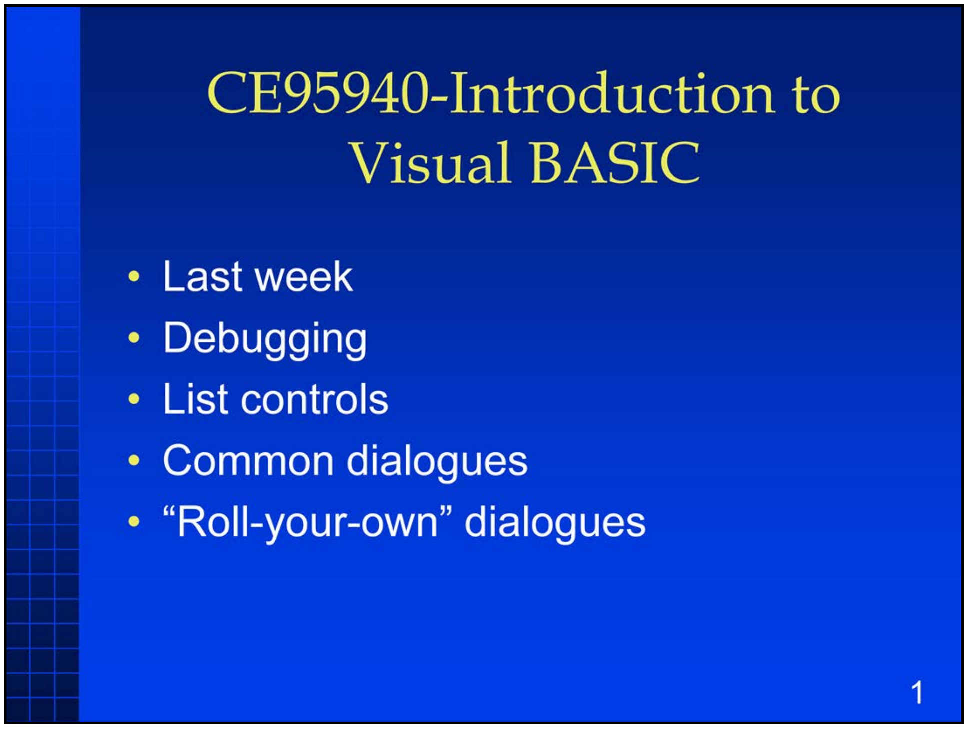
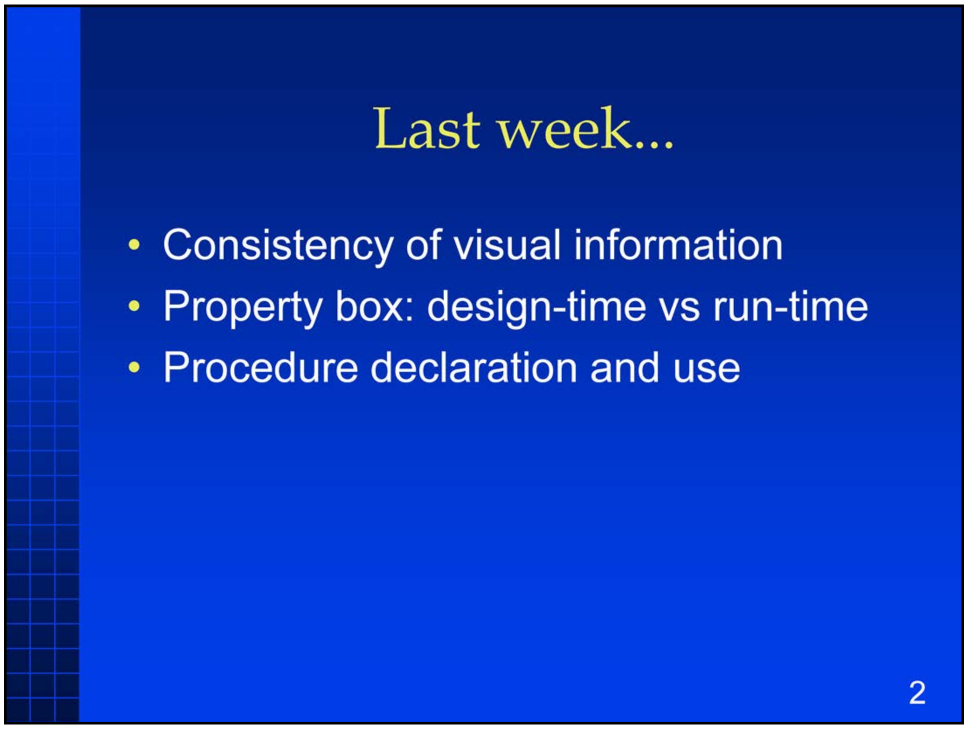


CE95940-Introduction to Visual BASIC

- Last week
- Debugging
- List controls
- Common dialogues
- “Roll-your-own” dialogues

Review common questions and problems from last week



Last week...

- Consistency of visual information
- Property box: design-time vs run-time
- Procedure declaration and use

2

important idea in exercises: consistency of visual information. If information is presented more than once, should be consistent. Will see more today with filename displays

be aware that prop. box only shows design-time props., use help for full details. Review use of F1 help, show TOC discovery path

review procedure declaration and use

Sometimes:

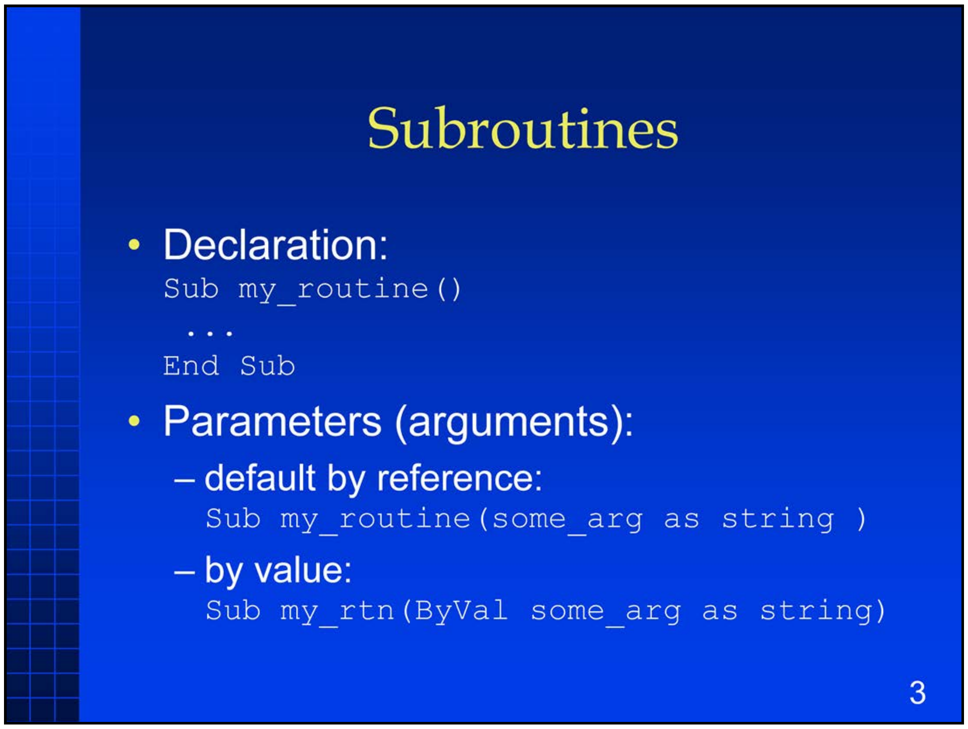
•“Opaque” vs “transparent” backgrounds

shape control has backstyle property -- allows background to be seen (opaque) or object behind shape to be seen (transparent)

•“Help” information for controls: F1 vs TOC

can use F1 for context-help at any time. Eg information about properties. Beware that property lists may omit run-time only properties.

Demonstrate use of help for controls.



Subroutines

- Declaration:

```
Sub my_routine()
  ...
End Sub
```

- Parameters (arguments):
  - default by reference:

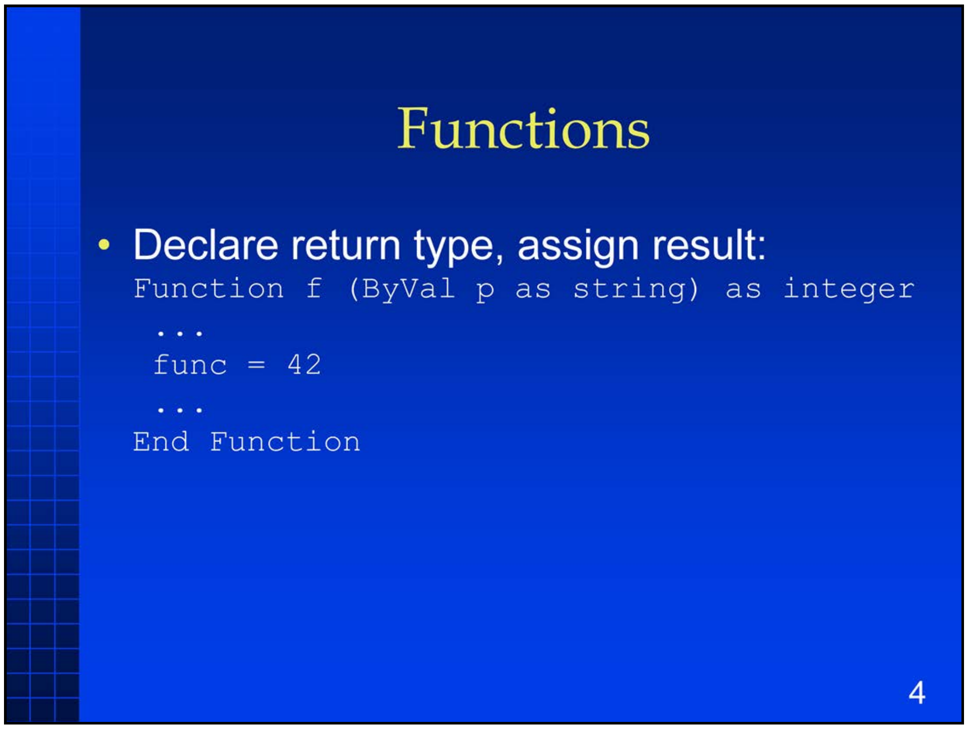
    ```
    Sub my_routine(some_arg as string )
    ```
  - by value:

    ```
    Sub my_rtn(ByVal some_arg as string)
    ```

3

terminology: procedure, subroutines, functions, prefer proc=no ret; func=ret, etc

defined with View menu, “New Procedure”

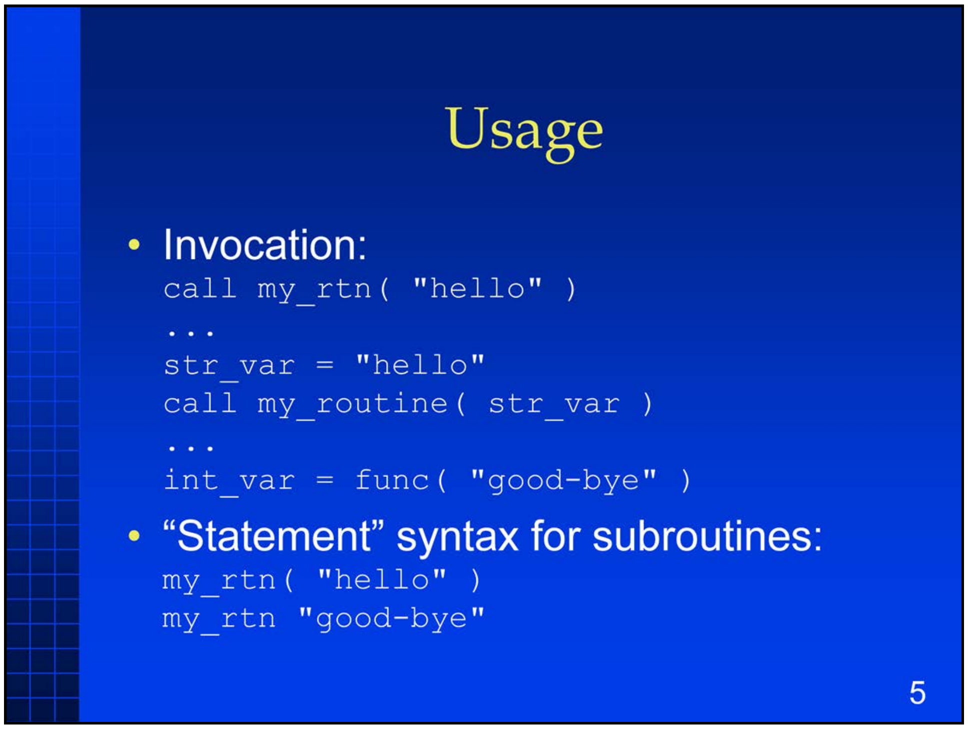


Functions

- Declare return type, assign result:

```
Function f (ByVal p as string) as integer
 ...
 func = 42
 ...
End Function
```

declare type of value to be returned



Usage

- Invocation:

```
call my_rtn( "hello" )
...
str_var = "hello"
call my_routine( str_var )
...
int_var = func( "good-bye" )
```

- “Statement” syntax for subroutines:

```
my_rtn( "hello" )
my_rtn "good-bye"
```

5

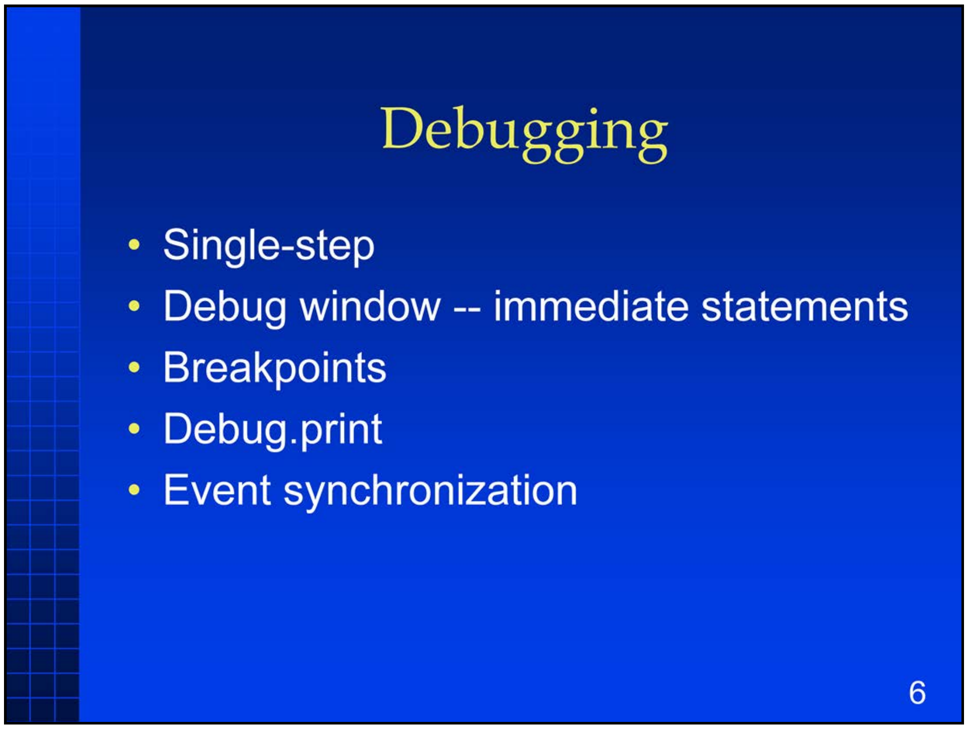
subroutines use “call”, functions used in expressions
reference args cannot be passed literals

alternate syntax for subroutines -- looks like built-in statement

recursion is permitted

function results cannot be discarded; must be assigned to something

VB check argument counts, does not check function result assignment



Debugging

- Single-step
- Debug window -- immediate statements
- Breakpoints
- Debug.print
- Event synchronization

6

start a program in single step

executing one statement at a time

stepping over procedure/function calls - only applies to user-written code

immediate: displaying with “?”; setting new values

stopping at arbitrary places with breakpoints

displaying values during execution: debug.print

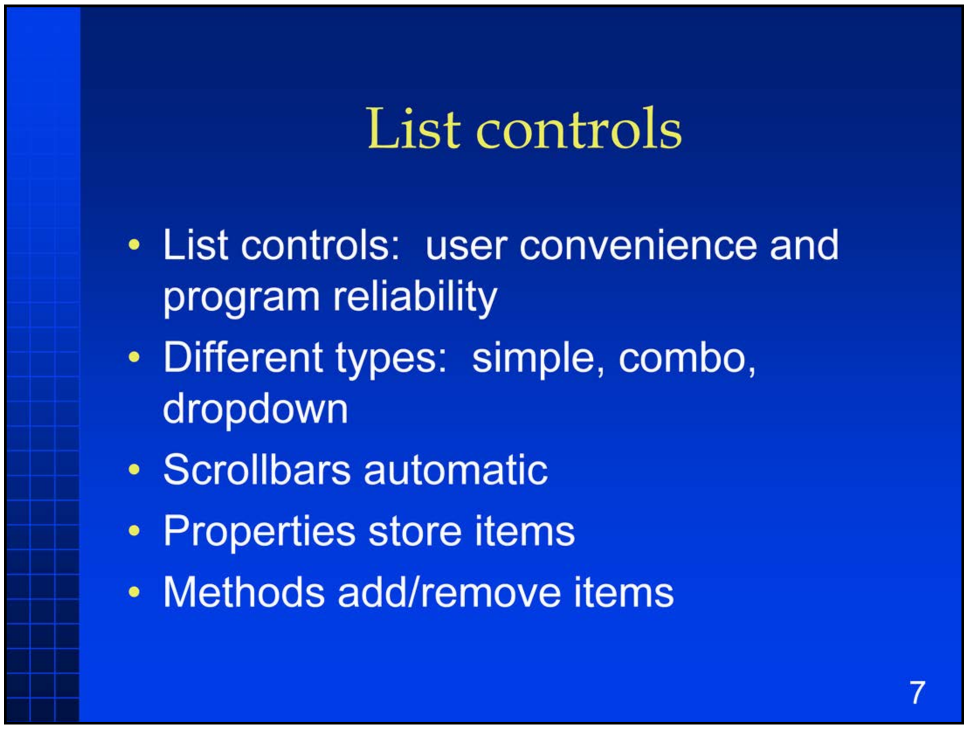
waiting for events?

create form, button, label, global integer var, dummy proc

form-load set caption

button calls proc,

proc increments global, sets label caption



List controls

- List controls: user convenience and program reliability
- Different types: simple, combo, dropdown
- Scrollbars automatic
- Properties store items
- Methods add/remove items

7

lists user to select one (or more) items from predetermined set of items

easier for the user; more secure for the developer (restricts range of input, simplifies input verification)

various types: simple, dropdown, combo, dropdown-combo

combo's are a combination of list and textbox -- allow selection from list or entry of text

list features: single or multiple selection, editing assists (ctrl, swipes)

combo features: cut, copy, paste shift-del, shift-ins, ctrl-ins; move-to-matching-item

vertical scroll-bars automatically added when necessary; can have horizontal scrolling with multi-column

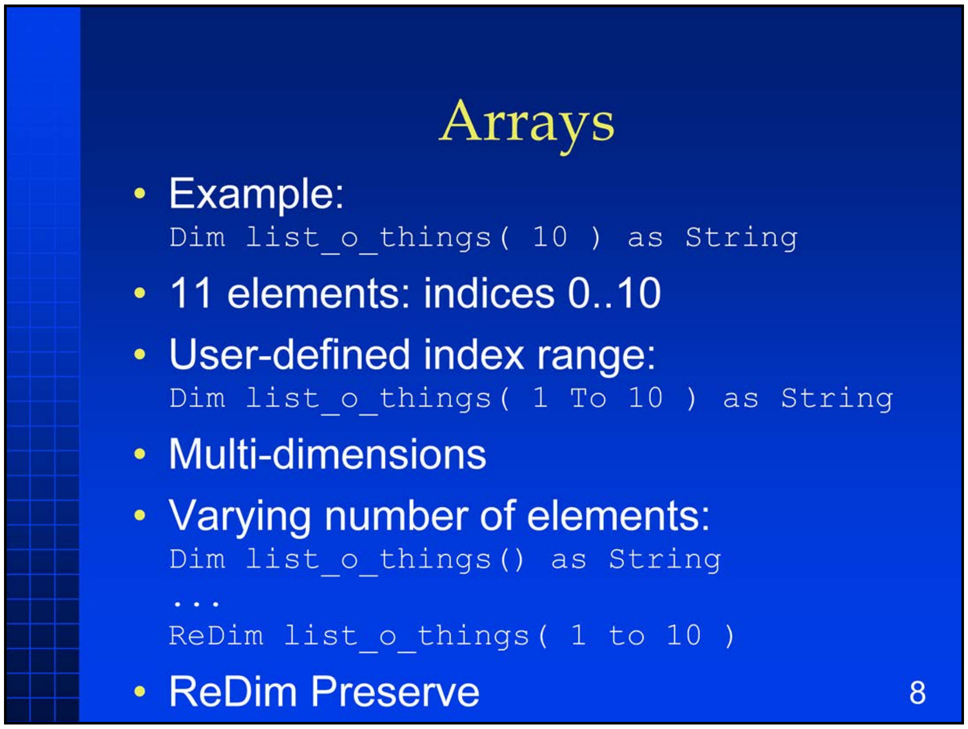
various properties used to control lists: style, sorted, number items, currently-selected

array property List contains the items in the list -- operates as an ordinary array

list maintained (items added or removed, list cleared) using methods

click, change event

Demo to follow...



Arrays

- Example:
```
Dim list_o_things( 10 ) as String
```
- 11 elements: indices 0..10
- User-defined index range:
```
Dim list_o_things( 1 To 10 ) as String
```
- Multi-dimensions
- Varying number of elements:
```
Dim list_o_things() as String
...
ReDim list_o_things( 1 to 10 )
```
- ReDim Preserve

similar to other programming languages

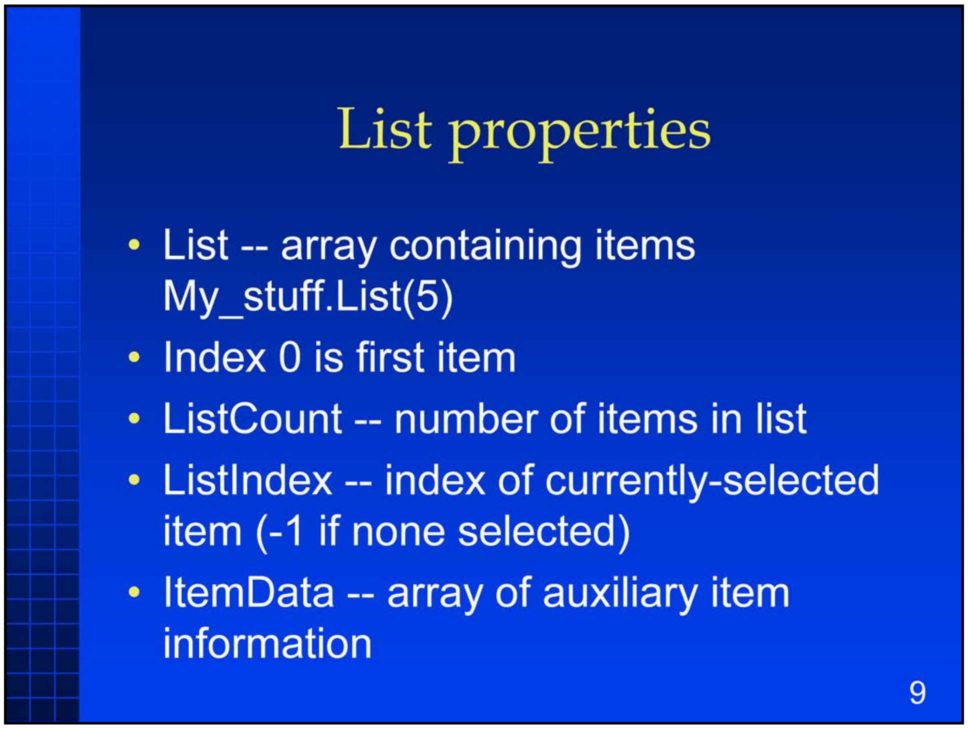
by default: zero-origin, so 11 elements

user-defined indices

matrices and higher-order structures

varying number of elements: declared with no elements, then redeclared later when number of elements changes

redimension an existing array and keep current contents: redim preserve

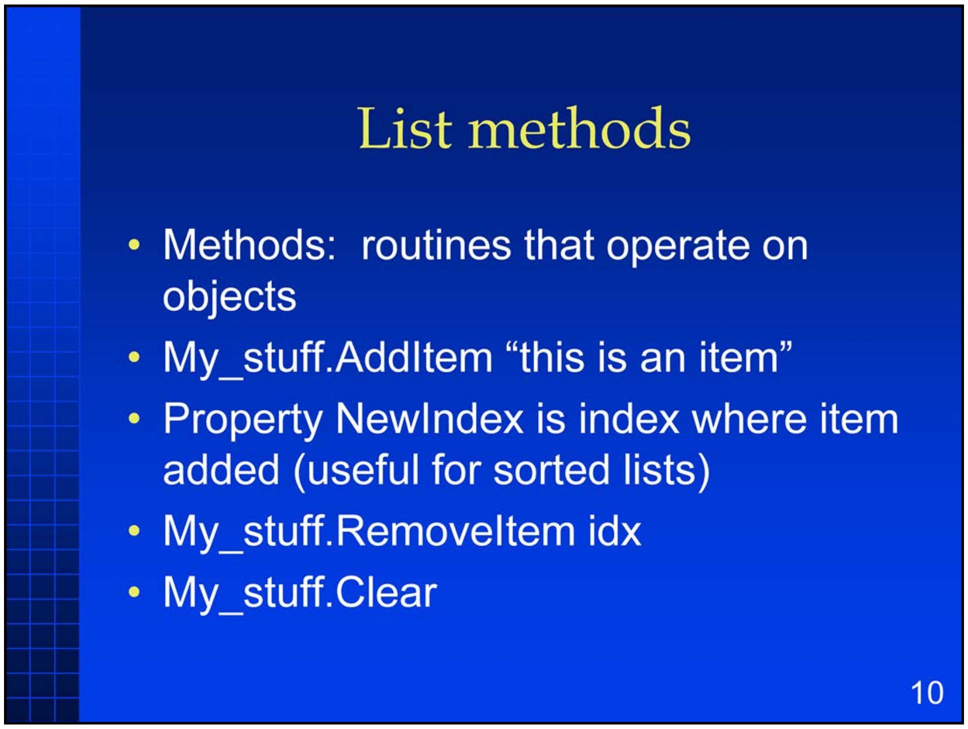


List properties

- List -- array containing items My_stuff.List(5)
- Index 0 is first item
- ListCount -- number of items in list
- ListIndex -- index of currently-selected item (-1 if none selected)
- ItemData -- array of auxiliary item information

9

example program pending



List methods

- Methods: routines that operate on objects
- My_stuff.AddItem “this is an item”
- Property NewIndex is index where item added (useful for sorted lists)
- My_stuff.RemoveItem idx
- My_stuff.Clear

10

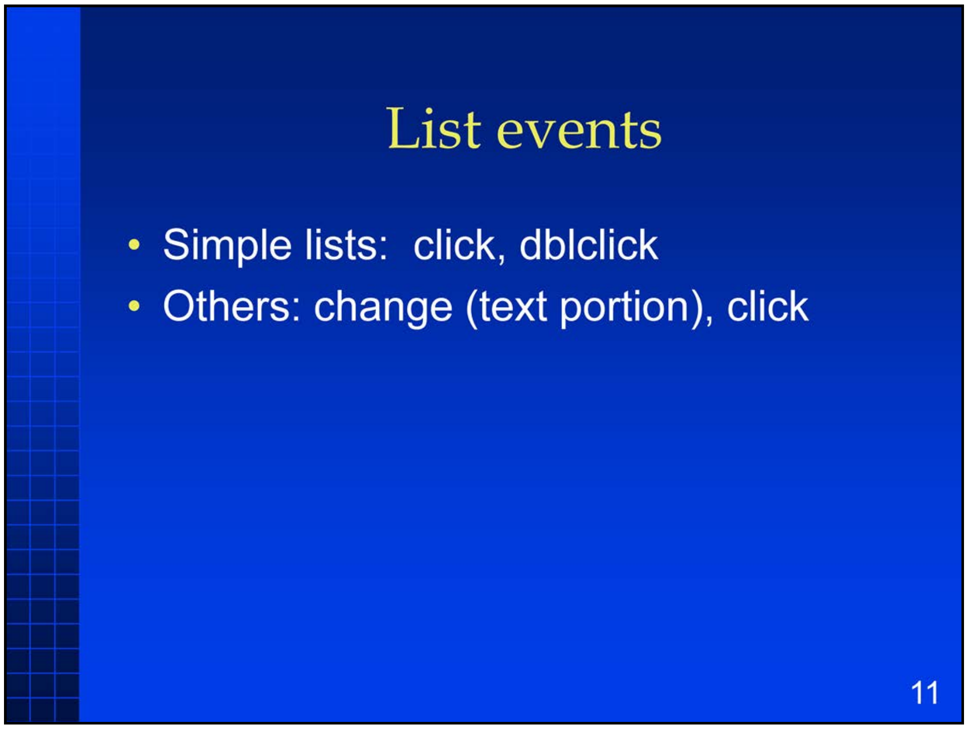
Methods: functions/procedures that are built-in to an object Can require arguments or not, can return values or not (executesql)

adding an item: item is a string. can associate numeric data using ItemData:

    my_stuff.ItemData(my_stuff.NewIndex)

remove an item, given a string

clear the list: remove all items



List events

- Simple lists: click, dblclick
- Others: change (text portion), click

11

for simple list control, click occurs as move up/down (dblclick as appropriate).

user click to update other fields

combo & dd-combo: change occurs when text changes (not necessarily click); click occurs with up/down

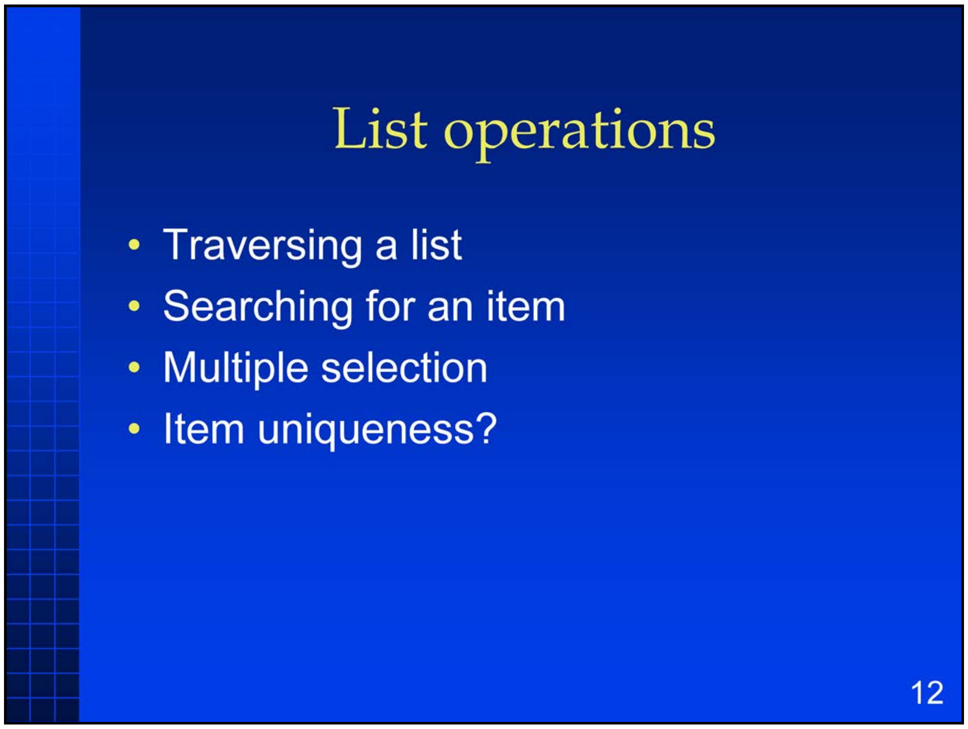
Example program: 3\1; listdemo

show operation

show load proc

demo delete

comment in debug stmts in slist.click & scombo



List operations

- Traversing a list
- Searching for an item
- Multiple selection
- Item uniqueness?

high-level functions not built-in

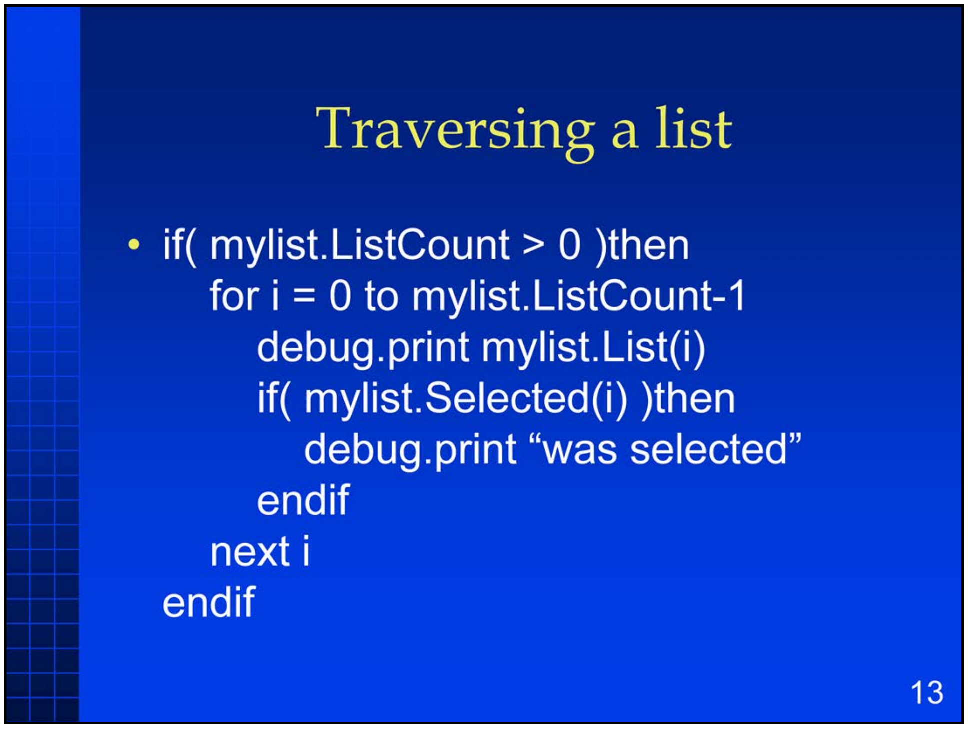
traversing == starting at the beginning and looking at each one

lots of code do

multiple selection -- simple lists only

use array property Selected - true if item selected otherwise false

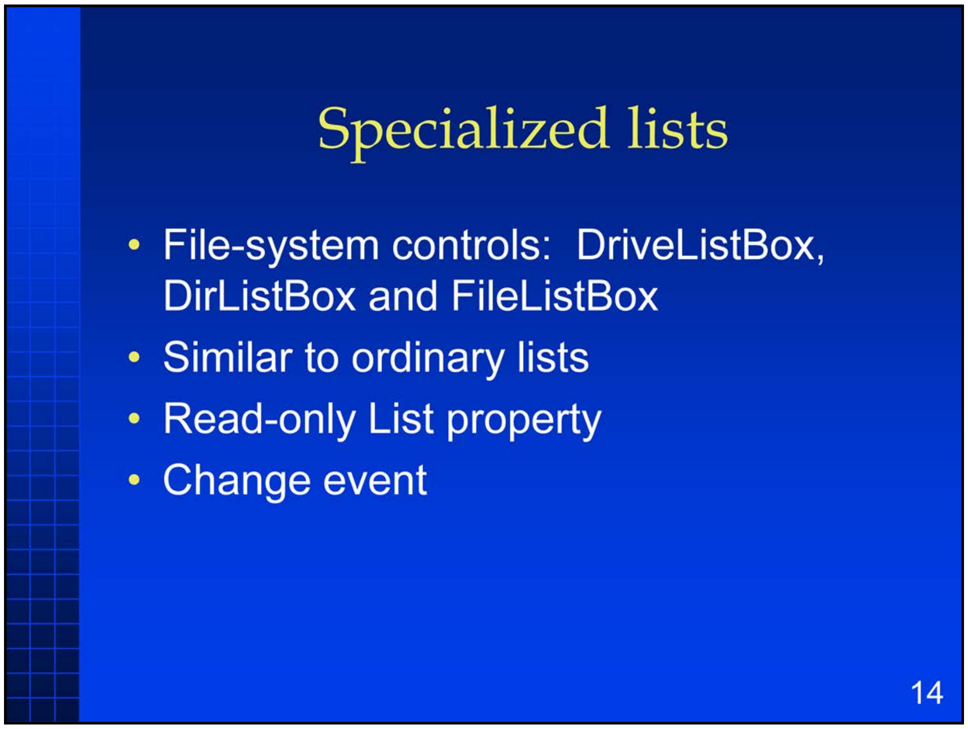
no guarantees of uniqueness -- programmer must handle



Traversing a list

```
• if( mylist.ListCount > 0 )then
     for i = 0 to mylist.ListCount-1
        debug.print mylist.List(i)
        if( mylist.Selected(i) )then
           debug.print “was selected”
        endif
     next i
  endif
```

assuming MultiSelect is set



Specialized lists

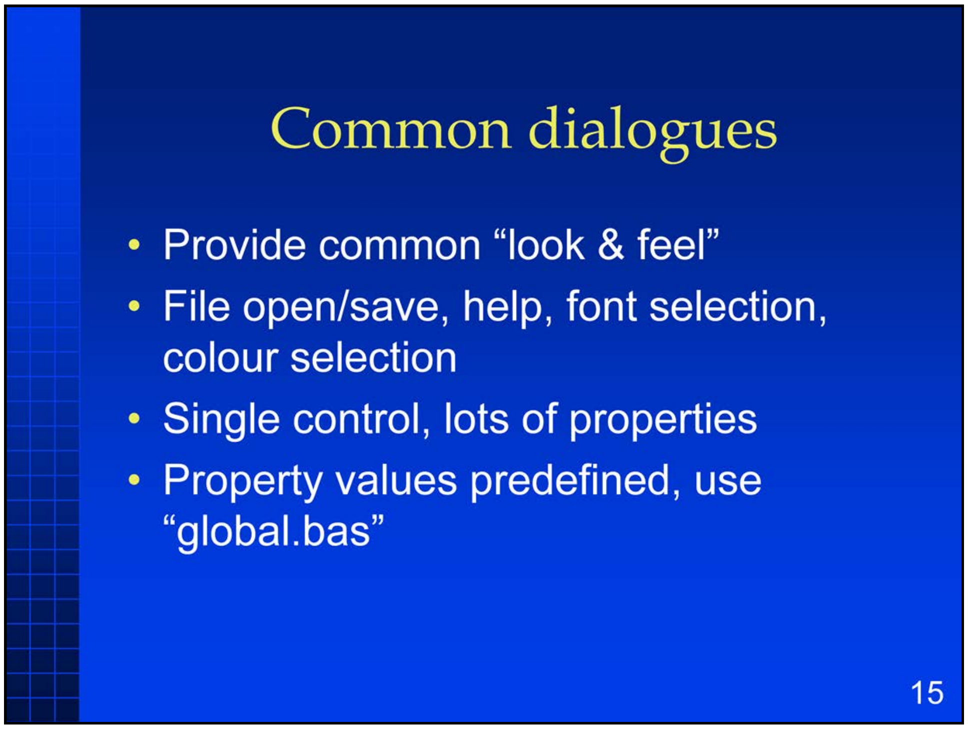
- File-system controls: DriveListBox, DirListBox and FileListBox
- Similar to ordinary lists
- Read-only List property
- Change event

14

show example program: 3\2; fs-ctrls

note consistency problem

Ok in some cases, prefer to have a traditional functional interface: call a function and have name returned



Common dialogues

- Provide common "look & feel"
- File open/save, help, font selection, colour selection
- Single control, lots of properties
- Property values predefined, use "global.bas"

15

Common dialogues: filename (open, save), help, colours, fonts

provide common appearance and operation for all applications

Single object with lots or properties

Property values predefined: use global.bas from constant.txt

Example: build a simple one showing the global.bas creation

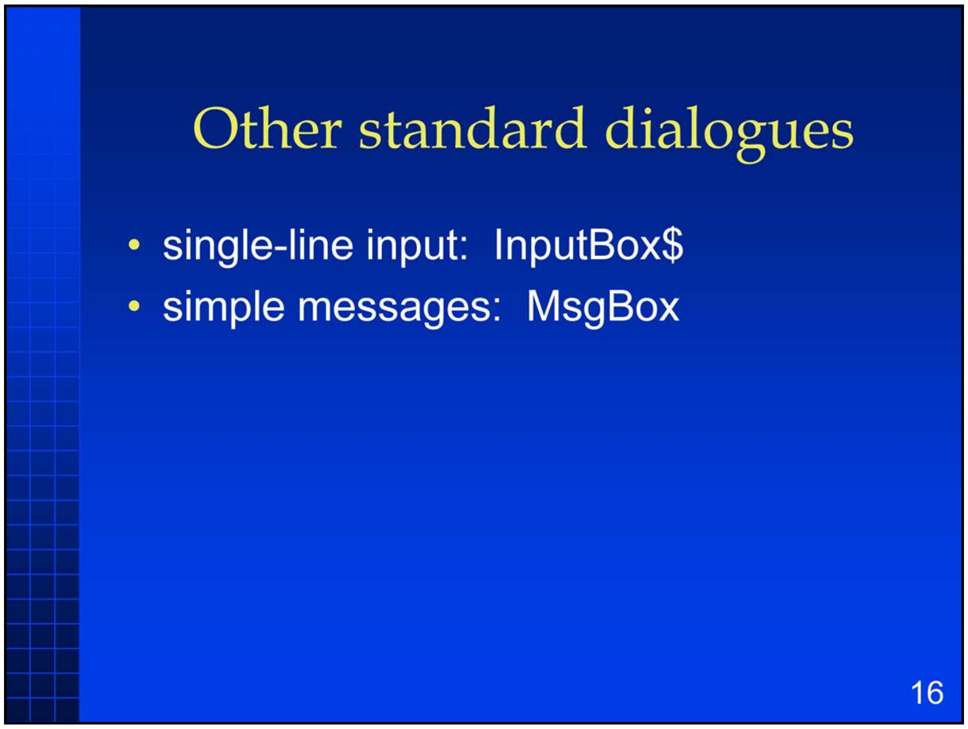
create form, place CD

add new module, save as global.bas

open constant.txt, find CD stuff

cut & paste

Example: 3\3; commdlg: switch to predefined one and demonstrate; look at code: emphasize activation via setting action property



Other standard dialogues

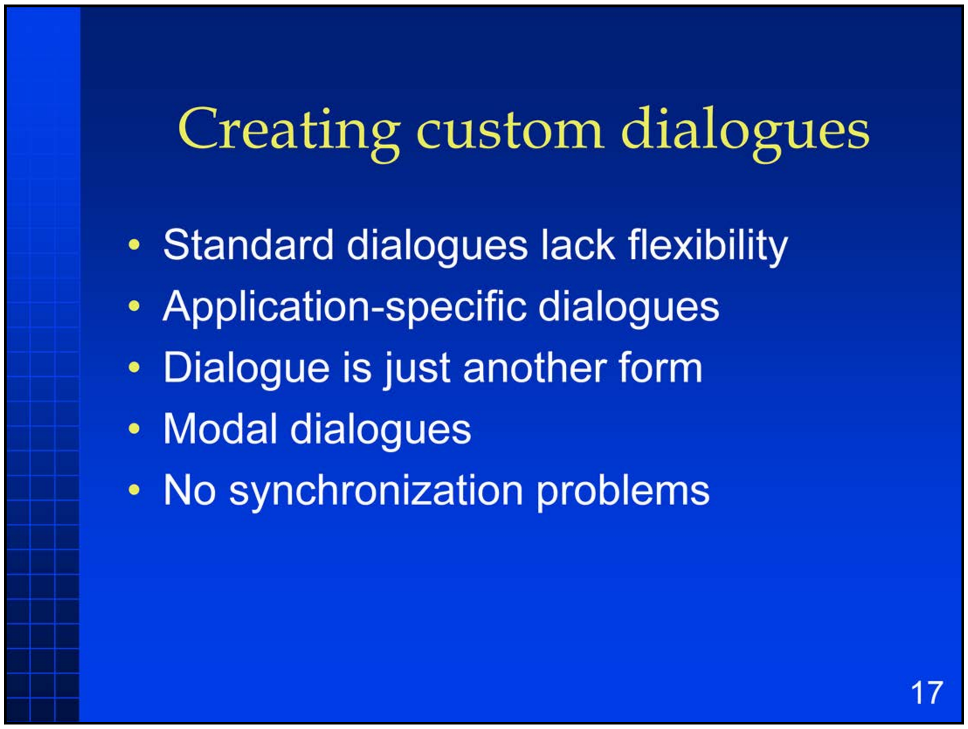
- single-line input: InputBox$
- simple messages: MsgBox

Quick, but inflexible (limited size of input etc)

implemented with functions instead of object -- reason unknown!

show demo 3\4; msgbox

show button click, show global.bas



Creating custom dialogues

- Standard dialogues lack flexibility
- Application-specific dialogues
- Dialogue is just another form
- Modal dialogues
- No synchronization problems

17

dialogues seen thus far are OK, but may not be enough -- want to create our own

a dialogue is just another form

we have seem “modal dialogues” that disable other parts of application

opposite is modeless dialogues, also called modeless forms, allow many parts of application to be active at the same time

modeless can create problems; if one form shows information that appears elsewhere, updates not automatic

modal prevents synchronization problems; restricts ui when appropriate

parameter passing not good, must use global variables

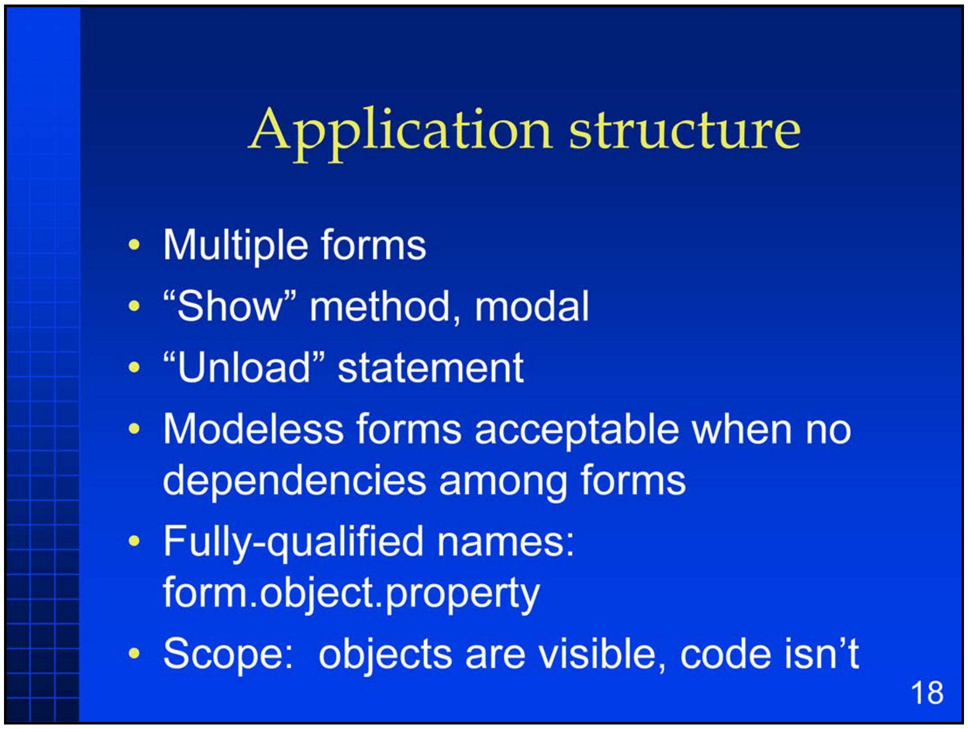
construct demo program of stubs (msgboxes etc)

form1 shows form2 modeless (default)

explain show method, add modal (value 1)

demo 3\5; modal; explain unload statement

comment out modal, run, explain!!



Application structure

- Multiple forms
- “Show” method, modal
- “Unload” statement
- Modeless forms acceptable when no dependencies among forms
- Fully-qualified names: form.object.property
- Scope: objects are visible, code isn’t

VB considers that all forms are present all the time, just can’t be seen.

Show makes them visible (equivalent to setting visible to true)

unload removes a form (opposite is load, but not generally needed -- if not loaded when references, loaded automatically)

Names of entities (properties, methods) need to be qualified if referred to from another form

Objects are visible outside a form, but code is not. If code is to be used in more than one form, must be placed into a code-only module (like global.bas). Data-sharing occurs with global variables.