LabVIEW Core 1

Instructor Guide

November 2014

Table of Contents

LabVIEW Core 1 Course Overview

Course Objectives

Given the LabVIEW environment, the learner will use LabVIEW best practices to implement a Simple State Machine application.

Target Audience

- New LabVIEW users
- Engineers and scientists
- Widely varied disciplines and applications
- Computer literate
- Very little past experience with LabVIEW
- Might be interested in certification
- Interested more in practice than assessment
- Both US and International

About This Guide

This Instructor Guide is your primary tool to plan, prepare for, and conduct the course. It is important for you to review this guide thoroughly, well in advance of the course. The following icons are included in this guide.

Icon	Definition and Use
📄	**Section** – Section header.
🖥	**Demo**– Demo this concept in the software.
📘	**Guide –** Refer participants to their participant materials*.*
📌	**Key Point** – Be certain to mention the key points listed.
🧑‍💻	**LabVIEW Exercise** – Students working in the software.
▶	**Multimedia** – Direct students to the multimedia file location: <Exercises>\LabVIEW Core 1\Multimedia\
?	**Question** – Questions to ask participants.
👥	**Group exercise** – Usually a discussion or activity for the whole class.
🧠	**Activity -** Knowledge checks and short exercises to help reinforce concepts.

Instructor Notes

You can find the most comprehensive and up-to-date information you need for this course on the NI Instructors community page: https://decibel.ni.com/content/groups/ni-instructors

Overview of Lessons

Lesson	Objective	Topics	Time (h:min)
Welcome and housekeeping			0:15
1 Navigating LabVIEW	Recognize the main components of the LabVIEW environment and be able to create a new project and a new VI in LabVIEW.	A. What is LabVIEW? B. Project explorer C. Parts of a VI D. Front panel E. Block diagram F. Searching for Controls, Functions, & VIs	1:30
2 Creating Your First Application (VI)	Use Express VIs to produce a project and simple VI that acquires and analyzes data and then display/s the results.	A. Dataflow B. LabVIEW data types C. Tools for programming, cleaning and organizing VIs D. Building a basic VI	3:00
3 Troubleshooting and Debugging	Fix a broken VI, debug incorrect results and behavior of a running VI, and display errors generated by functions while a VI is running.	A. Correcting broken VIs B. Debugging techniques C. Undefined or unexpected data D. Error handling	1:30
4 Using Loops	Recognize the different components of LV loop structures and apply a For or While appropriately given a scenario.	A. While Loops B. For Loops C. Timing a VI D. Feedback in Loops E. Plotting a waveform chart	2:30
5 Creating and Leveraging Data Structures	Create, manipulate and uses arrays, clusters, and type definition controls for data access and analysis.	A. Arrays B. Polymorphism C. Auto-indexing D. Clusters E. Type Definitions	4:20
6 Using Decision-Making Structures	Recognize the different decision-making structures and apply a Case Structure appropriately given a scenario with certain inputs, algorithm, and expected outputs.	A. Case Structures B. Event-driven programming	2:15
7 Modularity	Recognize the benefits of reusing code and will be able to create a subVI with a properly-configured connector pane, meaningful icon, documentation, and error handling.	A. Understanding Modularity B. SubVIs C. Icons and connector panes D. Using subVIs	2:10

Lesson	Objective	Topics	Time (h:min)
8 Acquiring Measurements with Hardware	Apply the open/iteratively perform action/close model to access and control a DAQ device.	A. Taking measurements with B. Acquiring measurements from NI DAQ hardware C. Acquiring measurements from non-NI instruments	1:40
9 Accessing Files in LabVIEW	Describe the basic concept of file I/O and apply the appropriate functions to a given scenario.	A. Accessing files from LabVIEW B. File I/O functions C. High- and low-level I/O D. File formats	1:00
10 Using Sequential and State Machine Programming	Recognize the benefits of sequential and state-based algorithms and apply techniques in LabVIEW to enforce sequential or state execution.	A. Using sequential programming B. Using state programming C. State machines	1:40
Next Steps			0:05

Course Timing

Target is about 7 hours/day active class time.

The following table contains a breakdown of the overall lesson and non-lecture activity duration.

Day	Activity	Name	Timing Estimates (h:m)
Day 1	Welcome slides		0:15
	Lesson 1	Navigating LabVIEW	1:30
	Activity 1-1	Confidence Activity: Building a VI	0:10
	Demo	Using the Project Explorer and Creating a VI	0:10
	Demo	Components of a VI	0:05
	Activity 1-2	Select Front Panel Objects	0:10
	Demo	Context Help and LabVIEW Help	0:10
	Exercise 1-1	Exploring a VI	0:15
	Demo	Searching for Controls, Vis, and Functions	0:05
	Exercise 1-2	Locating Controls, Functions, and VIs	0:15
	Activity 1-3	Lesson Review	0:10
	Lesson 2	Creating Your First Application	3:00
	Activity 2-1	Exploring Dataflow	0:20
	Demo	Accessing Object Properties	0:05
	Multimedia	Mechanical Actions of Boolean Controls	0:20
	Multimedia	Numeric Data Representations	0:20
	Demo	Enums	0:10
	Demo	Programming Tools	0:05
	Demo	Wiring Tips	0:05
	Exercise 2-1	Selecting a Tool	0:20
	Demo	Making Code Readable	0:05
	Activity 2-2	Program Architecture for Simple AAV VI	0:10
	Exercise 2-2	Simple Acquire, Analyze, and Visualize VI	0:30
	Activity 2-3	Lesson Review	0:10
	Lesson 3	Troubleshooting and Debugging VIs	1:20
	Activity 3-1	Review Debugging Tools	0:15
	Exercise 3-1	Debugging	0:25
	Demo	Automatic vs. Manual Error Handling	0:20
	Activity 3-2	Lesson Review	0:15
Day 2	Lesson 4	Using Loops	2:30

Day	Activity	Name	Timing Estimates (h:m)
	Activity 4-1	While Loops vs. For Loops	0:15
	Exercise 4-1	Pass Data Through Tunnels	0:25
	Demo	Wait Chart VI	0:05
	Multimedia	Using Shift Registers	0:15
	Demo	Creating Shift Registers	0:05
	Exercise 4-2	Calculating Average Temperatures	0:15
	Exercise 4-3	Temperature Monitor VI—Plot Multiple Temperatures	0:25
	Activity 4-2	Lesson Review	0:05
	Lesson 5	Creating and Leveraging Data Structures	4:30
	Demo	Viewing Arrays	0:10
	Multimedia	Common Array Functions	0:10
	Activity 5-1	Using Array Functions	0:15
	Exercise 5-1	Manipulating Arrays	0:45
	Demo	Create a Cluster Control	0:20
	Demo	Creating a Cluster on the Block Diagram	0:15
	Exercise 5-2	Temperature Warnings VI—Clusters	0:30
	Demo	Difference between Control, Type Def, and Strict Type Def	0:10
	Demo	Creating and Identifying Type Definitions (Type Def)	0:05
	Exercise 5-3	Temperature Warnings VI—Type Definition	0:30
	Activity 5-2	Lesson Review	0:20
Day 3	Lesson 6	Using Decision Making Structures	2:15
	Activity 6-1	Case Structure Review	0:05
	Demo	Case Structures	0:15
	Demo	Selector Terminal Types and Tunnels	0:10
	Exercise 6-1	Temperature Warnings with Error Handling	0:25
	Demo	Event-Driven Scenario	0:10
	Multimedia	Event-Driven Programming	0:15
	Demo	Configure and Use Events	0:10
	Exercise 6-2	Converting a Polling Design to an Event Structure Design	0:45
	Activity 6-2	Lesson Review	0:05
	Lesson 7	Modularity	2:15

Day	Activity	Name	Timing Estimates (h:m)
	Demo	Creating an Icon	0:05
	Exercise 7-1	Temperature Warnings VI—As SubVI	0:50
	Activity 7-1	Lesson Review	0:05
	Lesson 8	Acquiring Measurements from Hardware	1:40
	Multimedia	Measurement Fundamentals with NI DAQ	0:15
	Exercise 8-1	Using NI MAX to Examine a DAQ Device	0:20
	Exercise 8-2	Programming with the DAQmx API	0:30
	Multimedia	Automating Non-NI Instruments	0:10
	Exercise 8-3	Instrument Configuration with NI MAX	0:15
	Exercise 8-4	Exploring Instrument Drivers	0:10
	Activity 8-1	Lesson Review	0:05
	Lesson 9	Accessing Files in LabVIEW	1:00
	Exercise 9-1	Exploring High-Level File I/O	0:20
	Exercise 9-2	Temperature Monitor VI—Logging Data	0:30
	Activity 9-1	Lesson Review	0:03
	Lesson 10	Using Sequential and State-Based Designs	1:40
	Multimedia	Building State Machines	0:15
	Exercise 10-1	Weather Station Project	1:00
	Demo	Simple State Machine Project Template	0:05
	Activity 10-1	Lesson Review	0:05
	Next Steps		0:05

Welcome and Housekeeping

Objective: Acclimate participants to the facility and set expectations for the course.

Time: 15 minutes

	Content/Narrative	Materials/ Duration
Slide 1	**National Instruments** Title slide	
Slide 2	**LabVIEW Core 1** Title slide	
Slide 3	**What You Need to Get Started** Review the hardware and software requirements to get the most out of the course.	
Slide 4	**File Locations** Tell the students where to find the exercise and multimedia files on disk.	
Slide 5	**Instructional Methods** Go over the instruction methods for the course.	
Slide 6	**Getting the Most out of this Course** Reassure everyone that mistakes are okay and that they will not break anything. When a student makes a mistake, use it as a learning opportunity. Our goal is to teach a student to be self-sufficient when the instructor is no longer available.	
Slide 7	**LabVIEW Certification**	
Slide 8	**Course Learning Map**	
Slide 9	**This course prepares you to:** Go over some of the objectives of the course.	

Lesson 1. Navigating LabVIEW

Objective: Recognize the main components of the LabVIEW environment and create a new project and VI.

This lesson contains the following topics:
A. What is LabVIEW?
B. Project Explorer
C. Parts of a VI
D. Front Panel
E. Block Diagram
F. Searching for Controls, VIs, and Functions

Duration: 1 hour, 30 minutes

	Content/Narrative	**Duration**
Slide 1	State the lesson objective. Briefly outline the topics covered in this lesson.	
Slide 2	**A. What is LabVIEW?** **Objective:** Recognize the benefits of LabVIEW.	
Slide 3	**Benefits of LabVIEW** Recap from GS that LV is a graphical programming environment. • Interfaces with wide variety of hardware • Scales across different targets and OSs • Provides built-in analysis libraries	
Slide 4	**LabVIEW Language Characteristics** Highlight key characteristics of LabVIEW in Core 1 and Core 2.	
Slide 5	**Activity 1-1:** Confidence Activity Build a VI Assess level of class by getting students to complete this activity: Build a VI to calculate the product of two numbers. If students have no LabVIEW experience and have not viewed the Getting Started content, consider converting this activity into a demonstration.	10 min

Slide 6

B. Project Explorer

Objective: Recognize the main components of the Project Explorer and explain its role in application building.

Slide 7

LabVIEW Files

- Review from Getting Started that the Project Explorer is where all LabVIEW programming starts.
    - Manage project files
    - Deploy programs to remote embedded targets like CompactRIO
    - Create stand-alone executables and installers for distribution
    - Distribute code to developers and create shared libraries
    - Integrate LabVIEW programs with source code control
    - Perforce and TortoiseSVN
- Using the image on this slide, review the parts of the project explorer: Project root, My Computer, Dependencies, Build Specifications
- In this class you will learn about three different LabVIEW files – LabVIEW projects, VIs, and custom controls.
- LabVIEW projects can also include non-LabVIEW file types. For example, you can include documentation files.

Slide 8

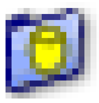
Demo: Using the Project Explorer and Creating a VI 10 min

Location: <Exercises>/Demonstrations/Using the Project Explorer/ Display Documentation.vi

1. Start with the Getting Started window and point out the Create Project button and the Open Existing button.
2. Create a blank project.
3. Show how to add:
    - A new, blank VI
    - An existing VI (Display Documentation.vi)
    - A text file (Weather Station Documentation.txt)
4. Show how to remove files from the project.
    - Be sure to demonstrate that changes made in the **Project Explorer** window on the **Items** tab do not reflect how the files are saved on disk.

Slide 9 **Adding Folders to a Project**

Highlight the difference between virtual folders and autopopulating folders.

Virtual folder	Auto-populating folder
• Organizes project • Does not represent files on disk	• Adds a directory on disk to the project • LabVIEW continuously monitors and updates the folder
	

Slide 10

C. Parts of a VI

Objective: Recognize and understand the difference between the front panel and block diagram.

Slide 11 **Demo: Components of a VI** 5 min

Location: <Exercises>/Demonstrations/Parts of a VI/Add and sub.vi

1. Open Parts of a VI.lvproj.
2. Open Add and Sub.vi
3. Review the parts of the VI.

Front Panel	User Interface Built with controls (inputs) and indicators (outputs)
Block Diagram	Contains the graphical source code Front panel objects appear as terminals on the block diagram
Icon/Connector pane	Necessary to use a VI as a subVI Graphical representation of a VI Map of the inputs and outputs of a VI

Slide 12

D. Front Panel

Objective: Recognize the components and functionality of the Front Panel window and select controls and indicators.

Slide 13

Front Panel Window Toolbar
Briefly mention the functionality of the items on the toolbar:

Run, Run Continuously, Abort Execution, Pause, Text Settings, Align Objects, Distribute Objects, Resize Objects, Reorder, Search, Help

Slide 14

Controls and Indicators
Briefly recap the difference between controls and indicators.

Controls	Indicators
Input	Output
Knobs, buttons, slides	Graphs, LEDs
Supply data to the block diagram	Display data the block diagram acquires or generates

Slide 15

String, Boolean, and Numeric Datatypes

- Recap the Boolean data type represents data that has only two options, such as True/False or On/Off
- The string data type is a sequence of ASCII characters. Use string controls to receive text from the user, such as a password or user name. Use string indicators to display text to the user.
- The numeric data in a control or indicator can represent numbers of various types, such as integer or floating-point.

Slide 16

Activity 1-2: Select Front Panel Objects 10 min

Goal: For each scenario, determine the appropriate data type and whether the front panel object should be a control or indicator.

Have students complete this activity, then come together as a class to review and discuss.

Slide 17

Front Panel Object Styles
LabVIEW has different control palettes with objects for building user interface, including the **Modern**, **Silver**, **Classic**, and **System** palettes.

The controls and indicators on the **Silver** palette are the newest to LabVIEW. They provide a rich user interface. Because of the added glyphs, the Silver objects tend to be a little larger than other styles.

Slide 18	**E. Block Diagram** **Objective:** Recognize features of the block diagram and be able to select functions.	
Slide 19	**Block Diagram Toolbar** Briefly mention the functionality of items on the toolbar. • Execution • Debugging • Editing	
Slide 20	**Demo: Components of a Block Diagram** **Location:** <Solutions>/Exercise 10-1/No HW/Weather Station.lvproj Show the following items on the block diagram: • Terminals ○ Demo how front panel objects appear as terminals on the block diagram ○ Demo viewing terminals as icons • Constants • Nodes ○ Functions ○ SubVIs ○ Structures • Wires • Free labels	10 min
Slide 21	**Express VIs** Briefly discuss visual appearance of Express VIs and how the learner configures these VIs in a configuration box. • Special type of subVI. • Require minimal wiring because you configure them with dialog boxes. • Save each configuration as a subVI. ○ This allows you to reuse the configuration without having to place and configure another instance of the Express VI. • Icons for Express VIs appear on the block diagram as icons surrounded by a blue field.	

Slide 22 **Node Appearance**

Highlight the differences in node appearance.

- Can display VIs and Express VIs as icons or as expandable nodes
- Expandable subVIs appear with a yellow field, and Express VIs appear with a blue field.
- Use icons if you want to conserve space on the block diagram.
- Use expandable nodes to make wiring easier and to aid in documenting block diagrams.
- By default, subVIs appear as icons on the block diagram, and Express VIs appear as expandable nodes.
- To display a subVI or Express VI as an expandable node, right-click the subVI or Express VI and remove the checkmark next to the View As Icon shortcut menu item.

Slide 23 **Wires**

Highlight how the wire color and size changes with the data type and dimension of the data that is being passed.

Slide 24 **Demo: Context Help and LabVIEW Help** 10 min

Location: <Solutions>/Exercise 10-1/No HW/Weather Station.lvproj

Recap Context Help and LabVIEW Help

- <Ctrl-H>
- Show how the Context help changes depending on the item that we hover over on the block diagram.
- Show how to show or hide Context Help
- Click the following button on the toolbar
- Show how to access the LabVIEW help:
  - Selecting Help»LabVIEW Help from the menu.
  - Clicking the Detailed help link in the Context Help window.
  - Right-clicking an object and selecting Help from the shortcut menu.

Slide 25 **Examples**

- Use the NI Example Finder to search all installed examples and examples located in the NI Developer Zone on the web.
- Modify any example VI to fit an application, or copy and paste from an example into a VI that you create.
- Show how to access the NI Example Finder.

Slide 26 **Exercise 1-1: Exploring a VI** 15 min

Goal: In this exercise, students identify the parts of an existing VI.

Open the Participant Guide and go over the Scenario of the exercise.

Have the students go through step 5 on their own and then go over the answers as a group. Then have the students finish the exercise on their own.

Slide 27 **Discussion: Exercise 1-1: Exploring a VI**

Question: What are constants and when should you use them?

Answer: Constants are terminals on the block diagram that supply fixed data values to the block diagram. Use constants when your VI needs to use the same value every time the VI runs.

Question: What are free labels and when should you use them?

Answer: Free labels are not attached to any object. You can create, move, rotate, or delete them independently.

Free labels are useful for documenting code on the block diagram and for listing user instructions on the front panel. Double-click an open space or use the Labeling tool to create free labels or to edit labels.

Slide 28

F. Searching for Controls, VIs, and Functions

Objective: Find the controls, indicators, and functions you need.

Slide 29

Demo: Searching for Controls, VIs, and Functions 5 min

Location: n/a – Start with a new VI.
Demonstrate searching for controls, VIs, and functions using the palettes, Quick Drop, and Global Search.

- **Controls Palette—**available only when the front panel window is the active window.
- **Functions Palette:**
  - Contains the VIs, functions, and constants you use to create the block diagram.
  - Navigate the subpalettes or use the Search button to search the Functions palette.
- **Quick Drop**: Press the <Ctrl-Space> keys to display.
- **Global Search:** Use the Search bar in the top right of the front panel and block diagram windows to search palettes, *LabVIEW Help*, and ni.com
- **Configuring palettes:**
  - Have students follow along as you show palettes and then configure to enable more visible palettes. By default, only the Express palette is visible.
  - Configure the following palettes to be visible:
    - Show the Silver controls palette on the Controls Palette
    - Show the Programming palette on the Functions palette.
  - Indicate that these changes persist on subsequent launches of LabVIEW.
- **Configuring front panel and block diagram settings:**
  - Open the Tools»Options dialog box, have students customize other common settings.
  - Front Panel page-Set Control Style for New VIs to Silver style
  - Block Diagram page-Uncheck the Place front panel terminals as icons checkbox. Configure Block Diagram Cleanup to customize your block diagram

Slide 30

Exercise 1-2: Locating Controls, Functions, and VIs 15 min
Goal: Use different ways to search for controls, functions and VIs.

This exercise walks the students through the different ways of searching and customizing the palettes.

Slide 31	**Discussion: Exercise 1-2: Locating Controls, Functions, and VIs** **Question:** Why would you want to add a function to the Favorites category in the Functions palette? **Answer**: You can put commonly-used functions in an easy-to-access category. **Question:** Why would you use Quick Drop instead of the Search button on the Controls and Functions palettes? **Answer**: You use Quick Drop for increased speed and the ability to create and leverage keyboard shortcuts. Refer to the *Quick Drop Keyboard Shortcuts* topic of the *LabVIEW Help* for more information on the built-in keyboard shortcuts and how to configure your own shortcuts.	
Slide 32 to 34	**Activity 1-3: Lesson Review** Have students complete the review at the end of the lesson in their Participant Guides. Then come together as a class to review and discuss. 1. Front Panel, block diagram, and Icon/Connector pane are the three parts of a VI.	10 min

Lesson 2. Creating Your First Application

Objective: Use Express VIs to produce a project and simple VI that acquires and analyzes data and then displays the results.

This lesson contains the following topics:
A. Dataflow
B. LabVIEW Data Types
C. Tools for Programming, Cleaning, and Organizing Your VI
D. Building a Basic VI

Duration: 3 hours

	Content/Narrative	**Duration**
Slide 1	State the lesson objective. Briefly outline the topics covered in this lesson.	
Slide 2	**A. Dataflow** **Objective:** The learner will be able to recognize characteristics of dataflow on the block diagram window.	
Slide 3	**Key Points of Dataflow** **Question**: Who completed the Getting Started Module on dataflow? Review concept of data flow in LabVIEW. • Node executes only when all input data are available. • Node supplies data to the output terminals only when they finish executing. Compare LabVIEW data flow programming to control flow programming (Visual Basic, C++, JAVA, etc) “In LabVIEW, the **flow of data** rather than the sequential order of commands determines the execution order of block diagram elements. You can create block diagrams that have **simultaneous operations**.” **(Optional) Demo**: Open a VI and show students what is happening using execution highlighting. **Location:** <Exercises>/Exploring A VI/Seconds Breakdown.vi	3 min
Slides 4-8	**Activity 2-1: Exploring Dataflow (Group Exercise)** Students can follow along in their Participant Guide. Students should complete individually before walking through the slides. Images from the Participant guide appear in slides 5-8.	20 min

Slide 5 **Dataflow: Example A**

Question: Which node executes first?

Answer: Either the File Dialog function or the Simulate Signal Express VI can execute first.

Question: Which node executes last?

Answer: Simple Error Handler VI

Question: Can the TDMS – File Viewer.vi execute before the TDMS Close functions?

Answer: No, All the inputs must be available before the node can execute.

"A well-designed block diagram typically flows from left to right. This makes it easier to see the flow of data on the block diagram."

Slide 6 **Dataflow: Example B**

Question: Which Express VI executes last?

Answer: Either the Statistics Express VI or the Write to Measurement File Express VI executes last or they execute in parallel.

The DAQ Assistant Express VI cannot execute last because both the Statistics Express VI and the Write to Measurement File Express VI are dependent on the data signal from the output of the DAQ Assistant Express VI.

“It is possible to have simultaneous operations.”

Slide 7 **Dataflow: Example C**

Questions: Which Express VI executes last?

Answer: The Write to Measurement File executes last because of dependencies on the DAQ Assistant and Statistics Express VIs.

Slide 8 **Dataflow: Example D**

Question: Which Tone Measurements Express VI executes last?

Answer: Either one of the Tone Measurements Express VIs can execute last.

Even though the Tone Measurements 2 Express VI has an extra dependency on the Filter Express VI, the Filter Express VI might execute before the Tone Measurements 1 Express VI allowing the Tone Measurements 2 Express VI to execute before the Tone Measurements 1 Express VI.

Slides 9	**B. LabVIEW Data Types** **Objectives:** The learner will be able to recognize the different data types and how they relate to front panel objects.	
Slide 10	**Terminals and LabVIEW Data Types** Review terminal colors, text, arrow direction, and border thickness. Have the students point out the difference between: • control and indicator terminals • terminal/wire colors • data type indications	
Slide 11	**Demo: Accessing Object Properties** **Location:** n/a – Start with a new VI. • Drop a control and indicator on front panel and show block diagram terminal. • Show how to access **shortcut menus**. “All LabVIEW objects have associated shortcut menus.” “Use shortcut menu items to change the look or behavior of objects.” • Show how to access the **Properties** dialog box. • Show how to multi-select and configure multiple objects through the Properties dialog box. For example, show and hide labels for multiple controls.	5 min
Slide 12	**Boolean Data** Review: • Booleans have only two values: True/False, On/Off, High/Low • Boolean control and indicator types: various Switches, Buttons, LEDs • What are some real world examples of Booleans?	
Slide 13	**Multimedia module:** *Mechanical Actions of Boolean* **Location:** <Exercises>\LabVIEW Core 1\Multimedia\Mechanical Actions of Boolean Controls **Intro:** Briefly talk about how different kinds of switches in the real world have different behaviors and mechanical actions depending on their application. Examples: Light switches—change state when the switch is flipped. The state stays the same until you flip the switch again. Buzzers and door bells—change state on a button press. They change back when the button is released. “The following module will introduce you to how Boolean controls can be configured for various mechanical actions."	20 min

Slide 14	**Multimedia module:** *Numeric Data Representations* **Location:** <Exercises>\LabVIEW Core 1\Multimedia\ Numeric Data Representations **Intro:** Briefly mention that LabVIEW supports several numeric formats and the following module introduces these representations in detail.	20 min
Slide 15	**Strings** • Describe strings: o A string is a sequence of ASCII characters o Various display styles: Backslash codes, Password, Hex • Describe special character constants found on String palette. • Describe uses for strings: o Creating simple text messages. o Instructing or prompting the user with dialog boxes. o Storing numeric data to disk in an ASCII file.	
Slide 16	**Demo**: **Enums** **Location:** n/a – Start with a new VI. Create an enum and add 3 items to show ability to rearrange/edit. • Describe enums: o Enums give users a list of items from which to select. o Enums make strings equivalent to numbers. o Numbers are easier than strings to manipulate on the block diagram • Show that each item represents a pair of values: String/16-bit integer • Point out that the data type of the enum terminal is blue, showing that the enum is passing an integer value.	10 min

Slide 17 **Other Data Types**

Describe other data types:

- Dynamic-Stores the information generated or acquired by an Express VI
- Path-Stores the location of a file or directory using the standard syntax for the platform you are using.
- Waveform-Carries the data start time, and dt of a waveform.

Question: How would a Path data type differ from a String data type?

Answer: Using the Path data type, LabVIEW can handle the folder syntax for the platform. (For example, a backslash on Windows)

Optional Demo: Finding help on data types

Location: n/a – Start with a new VI.

- Show context help.
- Have students following along as you show the *Block Diagram Objects* help topic in the LabVIEW Help.
  - Point out the default values column.
  - Suggest that the student learn the default value of common data types (numeric, string, Boolean)

3 min

Slide 18

C. Tools for Programming, Cleaning, and Organizing Your VI

Objective: The learner will recognize tools for making clean, readable front panels and block diagrams.

Slide 19 **Demo: Programming Tools** 5 min

Location: n/a – Start with a new VI.
Demonstrate several of the following:

- How to select an item before moving, cloning or deleting it.
- How to select the label and move it independently of the item.
- How to move a terminal using a drag operation.
- Cut/paste a terminal. Note the front panel object moves too.
- How to work with Boolean buttons. Change the Boolean label and the Boolean text using the Text tool.
    - "Label associates the control with the block diagram terminal."
    - "Boolean text is cosmetic and used only on the front panel."
- Resize a control (LED, Knob) using the resize handles.
- Show basic wiring of nodes… show ability to click, release, click
- Show how the cursor changes when hovering over a node terminal.
- Show how to change the color of a front panel control.
- Show that not all tools are available automatically. Select **View»Tools»Palette** to get to the Tools palette.

Slide 20 **Demo: Wiring Tips** 5 min

Location: <Exercises>/Demonstrations/Wiring Tips/

- Open Wiring Tips.lvproj.
- Open Add.vi
- Show how to use different diagram clean-up tools.
    - <Ctrl-B> to remove broken wires
    - Right-click a wire and select Clean Up Wire.
    - Clean-up Diagram button
- Show how to clone and move objects on the front panel and block diagram.

Slide 21 **Exercise 2-1: Selecting a Tool** 20 min
Goal: Practice resizing, moving, selecting objects, and wiring.

This is the first exercise spent entirely in LabVIEW. Spend extra time here if necessary until students are comfortable with the automatic tool selection.

Slide 22 **Discussion: Exercise 2-1: Selecting a Tool**

Question: How do you enable automatic tool selection?

Answer: Select View»Tools Palette and click the Auto button.

Slide 23 **Demo: Making Code Readable** 5 min

Location: n/a – Use blank VI.

- Show how to create free labels and owned labels.
- Show the difference between free labels and owned labels.

Owned Label	Free Label
Describe data content	Describe algorithms and add reference information
Have transparent background	Have yellow background
Move with object	Don’t move with the object

- Show how to attach a free label to an object
- Show how to set default values.

“Label structures to specify the main functionality.”

“Label long wires to identify their use/contents.”

“Label constants to specify the nature of the constant.”

Slide 24

D. Building a Basic VI

Objectives: The learner will recognize the Express VIs and be able to apply them appropriately.

Slide 25 **Building a Basic VI**

Review the Acquire-Analyze-Visualize paradigm.

- Acquire and output real-world signal with data acquisition and instrumentation devices
- Analyze data for meaningful information
- Share results using displays, reports, and the Web.

Express VIs are designed specifically for completing common, frequently used operations in each of these three task areas.

Slide 26	**Activity 2-2: Program Architecture for Simple AAV VI** Refer students to the Participant Guide to answer questions about the program architecture of a simple AAV VI and then discuss the answers as a group. **Program Architecture-Quiz Answers**: • Acquire: DAQ Assistant • Analyze: Statistics Express VI • Visualize – Log: Write to Measurement File Express VI • Visualize – Display: Waveform Graph indicator	10 min
Slide 27	**Exercise 2.2: Simple Acquire, Analyze, and Visualize VI** **Goal**: Create a simple VI that acquires data, analyzes data, and displays the results. Let students know that if they do not have DAQ hardware they can use the Simulate Signal Express VI to generate a waveform.	30 min
Slide 28	**Discussion: Exercise 2.2: Simple Acquire, Analyze, and Visualize VI** **Question**: How do you determine the file path of the generated text file? **Answer:** The file path of the text file is configured in the Write To Measurement File Express VI. Double-click the Write To Measurement File Express VI to find the file path of the text file.	
Slides 29-41	**Activity 2-3: Lesson Review** Refer students to the Participant Guide to answer the questions before showing the slides and discussing the answers with the group. 1. Which function executes first: Add or Subtract? **Add** 2. Which function executes first: Sine or Divide? **Divide** 3. Which of the following functions executes first: Random Number, Add or Divide? **Unknown** 4. Which of the following functions executes last: Random Number, Subtract or Add? **Subtract** 5. If an input to a function is marked with a red dot (known as a coercion dot), what does the dot indicate? **The value passed into a node was converted to a different representation.** 6. Which mechanical action causes a Boolean control in the FALSE state to change to TRUE when you click it and stay TRUE until LabVIEW has read the value? **Latch When Pressed**	10 min

Lesson 3. Troubleshooting and Debugging VIs

Objective: Fix a broken VI, debug incorrect results and behavior of a running VI, and display errors generated by functions while a VI is running.

This lesson contains the following topics:
A. Correcting Broken VIs
B. Debugging Techniques
C. Error Handling

Duration: 1 hour 30 min

	Content/Narrative	**Duration**
Slide 1	State the lesson objective. Briefly outline the topics covered in this lesson.	
Slide 2	**A. Correcting Broken VIs** **Objective:** Recognize an unexecutable VI and restate common problems.	
Slide 3	**Common Causes of Broken VIs** Remind students that the first few topics of this lesson will recap topics they learned in the Getting Started module, so they will not be covered in depth. **Question**: Who can identify the problems with the examples shown on the slide? • Mismatched data types results in broken wire • Two indicators wired together • Required input is unwired • SubVI is broken because of an unwired required input **Transition to next slide**: To find out why the VI is broken, click the broken run arrow. This launches the Error List Dialog.	
Slide 4	**Identify Problems and Fix Broken VIs** **Question:** Do you recognize this dialog box? How would you use it to debug your broken VI?	

Slide 5

B. Debugging Techniques

10 min

Objectives: Identify the LabVIEW tools for debugging and select the tool appropriate to the situation.

Slide 6

Debugging Tips

- Debugging techniques are also used when a VI produces unexpected data or behavior, not only when the VI has a broken Run arrow.
- The items on this slide are just a few of the debugging techniques you can use.
    - Is numeric representation correct?
    - Do nodes execute in the right order
    - Are there any unwired or hidden subVIs?
    - Does the VI pass undefined data?

Slide 7

Activity 3-1: Review Debugging Tools

15 min

Remind students that this is a recap of Getting Started. Don't spend too much time on it.

Have students complete the activity in the Participant Guide and then review the answers as a group.

Activity Answer

- "VI is returning unexpected data…" – **Probe** (this answer is provided in the Participant Guide as a sample)
- "You want to see how data moves…" – **Execution Highlighting**
- "You want to view each action…"–**Single-stepping**
- "This tool slows down how fast…" –**Execution Highlighting**
- "Your block diagram is complicated…"–**Probe**
- "Suspend the execution of a subVI…"–**Single-stepping**

Transition: These are all tools that you learned about in Getting Started. There are additional tools and techniques you can use to troubleshoot problems.

Slide 8

Breakpoints

This is new to students in this course.

- When you reach a breakpoint during execution, the VI pauses and the **Pause** button appears red.
- You can take the following actions at a breakpoint:
    - Single-step through execution using the single-stepping buttons.
    - Probe wires to check intermediate values.
    - Change values of front panel controls.
    - Click the **Pause** button to continue running to the next breakpoint or until the VI finishes running.
    - If students are familiar with SubVIs, mention "suspend when called": Pauses execution of a subVI when it is called.

Slide 9

Retain Wire Values

- Retain Wire Values enables you to probe wire values after execution completes. Normally, you would have to configure the probes before running the VI.
- This uses more memory since LV is storing data for each wire.
- Always remember to turn this feature off when you are done debugging.

Slide 10

Undefined or Unexpected Data
Check for unexpected Inf or NaN values in your data.

Slide shows example of how you might produce unexpected data.

Note that LabVIEW can process complex numbers using the functions on the Complex palette.

Slide 11

Exercise 3-1: Debugging 25 min
Goal: Use debugging tools and troubleshooting techniques to correct the problems of a broken VI.

Open the Participant Guide and review the scenario with students.

Slide 12

Discussion: **Exercise 3-1: Debugging**

Question: If you have a VI with a broken Run arrow, what should you do first?

Answer: Click the broken Run arrow to list the errors. Use the Error list window to locate the errors

Question: After you fixed your broken Run arrow, your VI results show unexpected data. What can you do next?

Answer: Use debugging tools such as Execution Highlighting, Single-stepping, Breakpoints, and Probes to debug your code.

Transition: Now we’ll look at ways to handle errors that occur when your VI is running.

Slide 13

C. Error Handling

Objectives: Describe the difference between automatic and manual error handling.

Slide 14

Error Handling Review
This is a review of Getting Started material

Questions:

- Can anyone explain what error handling is?
- Can anyone describe automatic error handling?
- Can anyone describe manual error handling?

Slide 15 **Demo: Automatic vs. Manual Error Handling** 20 min

Location: \LabVIEW 2014\examples\File IO\Spreadsheet\Tab-Delimited Data

Demonstrate the difference between automatic and manual error handling:

- Open a shipping example that includes a simple sequence of VIs. Try a file I/O example for this. File I/O examples tend to be relatively simple and chain VIs together using the error cluster wire.
- Verify that all error wires are wired correctly.
- Insert an error or cause an error during execution, such as cancelling an open File Dialog. Show that the Simple Error Handler displays the error.
- Check or enable automatic error handling in the VI Properties»Execution dialog box.
- Run the VI again. Show how automatic error handling highlights the node that caused the error.

NOTE: Simply enabling Automatic Error Handling does not override Manual Error Handling. If the error cluster is wired and the VI uses the Simple Error Handler, then LabVIEW defaults to manual error handling.

Slide 16 **Disable Automatic Error Handling**

Have participants follow along as you describe how to disable automatic error handling.

Explain that you might disable automatic error handling because although chaining error wires together effectively disables automatic error handling, you might forget to wire everything with error wires. To ensure that LabVIEW doesn’t resort to automatic error handling you should disable it for all new VIs.

Slide 17 **Error Clusters**

- Describe parts of error clusters.
  - **Status** is a Boolean value that reports TRUE if an error occurred. Most VIs, functions, and structures that accept Boolean data also recognize this parameter
  - **Code** is a 32-bit signed integer that identifies the error numerically. A non-zero error code coupled with a status of FALSE signals a warning rather than a fatal error.
  - **Source** is a string that identifies where the error occurred
- If LabVIEW detects an error, the node passes the error to the next node without executing that part of the code.

Slide 18 **Errors and Warnings**

- The Boolean control **Status** is TRUE (x) if an error has occurred and FALSE (checkmark) if no error has occurred.
- Although the **Code** for most error codes are negative and warning codes are positive, this is not universally true.
- Most products and VI groups produce only errors. Some products and VI groups can produce warnings. VISA is an example of a product group that can produce warnings.

Slide 19	**Merge Errors** Merge Errors function returns the first error found. If no error is found, it returns the first warning. “The Merge Errors function does not concatenate errors.”	
Slide 20	**Errors and Warnings Recommendations** Use the Simple Error Handler or another error reporting mechanism to report and display errors. OK Message is selected by default. This input determines the behavior of the Simple Error Handler dialog.	
Slide 21-29	**Activity3-2: Lesson Review** Refer students to the Participant Guide to answer the questions before showing the slides and discussing the answers with the group. 1. Which of the following will result in a broken run arrow? **A subVI is broken**, **A required subVI input is unwired**, **A Boolean terminal is wired to a numeric indicator** 2. Which of the following are the components and data types of the error cluster? **Status: Boolean, Code: 32-bit integer, Source: String** 3. All errors have negative error codes and all warnings have positive error codes. **FALSE** 4. Merge Errors function concatenates error information from multiple sources. **FALSE**	15 min

# Lesson 4.	Using Loops

Objective: Recognize the different components of LabVIEW loop structures and apply a For or While Loop appropriately for a given scenario.

This lesson contains the following topics:

A. Loops (Overview of Getting Started)
B. While Loops
C. For Loops
D. Timing a VI
E. Using Shift Registers (multimedia)
F. Plotting Data

Duration: 2 hours 30 min

	Content/Narrative	**Duration**
Slide 1	State the lesson objective. Briefly outline the topics covered in this lesson.	
Slide 2	**A. Loops Review** **Objectives:** Recognize loop structures and explain how to use them.	
Slide 3	**While Loops—Review** This material is review from Getting Started. Hit these high points. **While Loops** • Similar to a Do Loop or a Repeat Until Loop • Executes code until a condition is met o Conditional terminal – stop if true, continue if true • Iteration terminal o Returns number of times loop has executed o Zero-indexed • On Structures palette. Select and draw on block diagram.	
Slide 4	**For Loops - Review** Review from Getting Started. **For Loops** • Executes a subdiagram a set number of times • Count terminal indicates how many times to execute the code • Iteration terminal – same as While Loops • Like While Loop – on Structures palette – select and draw on block diagram	

Slide 5

For Loop / While Loop Comparison
Start with these questions directed at the examples shown on the slide.

Question: How many times will this For Loop Execute? What about the While Loop?

Answer: For Loop – 45, While Loop – Unknown

Then cover these points:

For Loop:
- Executes a set number of times
- Can execute zero times
- Tunnels automatically output array

While Loop
- Stops executing only if condition is met
- Must execute at least once
- Tunnels automatically output last value

Note: this information appears in a table in the Participant Guide.

Slide 6

Activity 4-1: **While Loops vs. For Loops** 15 min
Refer students to Participant Guide to complete the Activity, and then go over the answers with the class.

Slide 7

B. While Loops

Objective: Recognize tunnels and explain their purpose on a loop structure. Demonstrate how to use error checking and error handling inside a loop.

Slide 8

Tunnels
- Feed data into and out of a loop
- Same color as the data type
- Loop executes only after data arrive at the tunnel
- Data is passed out of the loop after the loop finishes executing

---

Slide 9

Error Checking and Error Handling:
- Wire error cluster to conditional terminal to stop the iteration of the loop
- Error cluster only passes Boolean value of the status parameter to the conditional terminal
- Loop stops on error

Slide 10

Exercise 4-1: Pass Data Through Tunnels

25 min

Goal: Use a While Loop and an iteration terminal to pass data through a tunnel.

Open the Participant Guide and go over the Scenario and Flowchart sections with the students.

Slide 11

Discussion: Exercise 4-1: Pass Data Through Tunnels

Question: How many times is the Number of Iterations indicator updated? Why?

Answer: Once, because the output tunnel on the While Loop will output only a single value once the While Loop has finished executing.

Slide 12

C. For Loops

Objectives: Demonstrate how to add a conditional terminal to a For Loop and describe how numeric conversion occurs on the For Loop count terminal.

Slide 13

Conditional Terminal

You can add a conditional terminal to configure a For Loop to stop when a Boolean condition is true or an error occurs.

For Loops configured with a conditional terminal have:

- A red glyph next to the count terminal.
- A conditional terminal in the lower right corner

Slide 14

Count Terminal Numeric Conversion

- The For Loop Count terminal always coerces to a 32-bit signed integer.
- For Loop can only execute an integer number of times.

Slide 15

D. Timing A VI

Objective: Identify scenarios that require loop timing and apply the appropriate function.

Slide 16

Why do you need timing in a VI?

- Control the frequency at which the loop executes
- Give the processor time to complete other tasks

Slide 17

Wait Functions Inside a Loop

Use wait functions inside a loop to allow the processor to address other tasks during the wait time.

- Wait (ms) – Waits the specified number of milliseconds.
- Wait Until Next ms Multiple – Waits until the value of the millisecond timer becomes a multiple of the specified value.
- Time Delay – This Express VI is a wrapper around the Wait (ms) function. This function includes error in and error out clusters for dataflow.

Slide 18 **Wait Function Timing**

- The X-axis represents the value of the system millisecond timer. For this timing chart, both timers are configured with input values of 100 and begin executing at time=50ms. The Wait Until Next ms Timer function finishes its 1st execution in 50ms, because 100 is the next multiple of 100. For the second loop iteration, it finishes at 200 because that is the next multiple of 100.
- The Wait (ms) function runs for 100ms each time, so it finishes at 150 and 250ms.
- This timing diagram assumes that the wait functions begin running immediately for each loop iteration and that the loop is ready to iterate as soon as the wait function finishes.

Slide 19 **Elapsed Time Express VI**

- Determines how much time elapses after some point in your VI.
- Keeps track of time while the VI continues to execute.
- Does not provide the processor with time to complete other tasks.

Slide 20 **Demo: Wait Chart VI** 5 min

Location: <Exercises>\Demonstrations\Wait Chart\Wait Chart.lvproj

Compare and contrast the Wait until Next ms Multiple function and the Elapsed Time Express VI for software timing.

Show block diagram of Wait Chart.vi and explain what's going to happen, then switch to front panel and run the VI.

Slide 21

E. Data Feedback in Loops

Objective: Apply shift registers when appropriate and predict the correct value at different iterations of the loop.

Slide 22 **Introduction to Shift Registers**

- Store values from one iteration to the next
- Right stores data on completion of an iteration
- Left provides stored data at beginning of next iteration

Slide 23

Multimedia Module: *Using Shift Registers* 15 min

Location: <Exercises>\LabVIEW Core 1\Multimedia\Using Shift Registers

Tell students what information is covered in the module
- Shift Registers – store data from previous loop iterations
- Initializing Shift Registers – determine what value the shift register starts with
- Default for Unwired Values – uses default on first run
- Compound Shift Registers – can store multiple previous iterations

Direct students to open and run the module.

Transition: Now that you've learned about shift registers, I'll show you how to create them and then you can practice using them in the next exercise.

Slide 24

Demo: Creating Shift Registers 5 min

Location: n/a
Create shift registers and convert existing tunnels to shift registers.
- Start with a Blank VI.
- Put a loop down, right-click and select **Add Shift Register**, and then wire data into it to show how it changes to the data type wired to it.
- Wire a terminal to the left side. Right-click the tunnel and select **Replace with Shift Register**. Point out that the second shift register is added to the right side.
- Wire another terminal all the way through the loop and then select one of the tunnels and **Replace with Shift Register**. Point out that they have to select the paired tunnel to complete this action.
- **New in 2014**: If you place a loop around a function that has matching input/output wires (for example, error in and error out), instead of creating tunnels for those wires, LabVIEW now creates shift registers.

Slide 25

Exercise 4-2: Calculating Average Temperature 15 min
Goal: Use a While Loop and shift registers to average data.

Open the Participant Guide and review the scenario and design with students.

Slide 26

Discussion : Exercise 4-2: Calculating Average Temperature

Question: You calculated the average of the last 5 temperature readings. How would you modify the VI to calculate the average of the last 10 temperature readings?

Answer: Resize the shift register in the VI to have five additional elements. Resize the compound arithmetic function to add 10 values and divide the result by 10.

Slide 27

F. Plotting Data—Waveform Chart

Objective: Use data feedback in a loop to plot waveform charts.

Slide 28

Waveform Chart

- Special type of numeric indicator
- Display single or multiple plots

Slide 29

Waveform Chart Properties

- Show and hide legends
- Change color and line styles
- Change interpolation styles

Slide 30

Exercise 4-3: Temperature Monitor VI—Plot Multiple Temperatures 25 min

Goal: Plot multiple data sets on a single waveform chart and customize the chart view.

Open the Participant Guide and review the scenario and design with students.

Slide 31

Discussion: Exercise 4-3: Temperature Monitor VI—Plot Multiple Temperatures

Question: In what ways do the following tools allow the user to interact with the plot?

- Plot Legend
- Graph Palette
- Scale Legend

Answer:

- **Plot Legend**-Customize how a plot appears in a chart. Ex. Line & point style, interpolation, color, fill
- **Graph Palette**-Panning tool, zoom
- **Scale Legend**-Autoscale on/on set scale format, precision, scale visibility, grid color

Slides 32-34

Activity 4-2: Lesson Review 5 min

Refer students to the Participant Guide to answer the questions before showing the slides and discussing the answers with the group.

1. Which structure must run at least one time? **While Loop**

Lesson 5. Creating and Leveraging Data Structures

Objective: Create, manipulate and use arrays, clusters, and type definition controls for data access and analysis.

This lesson contains the following topics:

A. Arrays
B. Common Array Functions
C. Polymorphism
D. Auto-Indexing
E. Clusters
F. Type Definitions

Duration: 4 hours 20 min

	Content/Narrative	**Duration**
Slide 1	State the lesson objective. Briefly outline the topics covered in this lesson.	
Slide 2	**A. Arrays** **Objective**: Identify when to use arrays and learn how to create and initialize arrays.	
Slide 3	**Arrays** Review from getting started the following: • What is an array • Why use arrays • Arrays are 0-indexed.	
Slide 4	**1D and 2D Examples** Recap from GS that arrays can be 1D or 2D. • Example of 1D: Collection of temperature data • Example of 2D: A table of data with rows and columns	
Slide 5	**2D Arrays** Resizing the index display to get the number of dimensions you need in the array. • Create a multidimensional array on the front panel by right-clicking the index display and selecting Add Dimension from the shortcut menu. • Resize the index display until you have as many dimensions as you want.	

Slide 6

Initializing Arrays

- By default all elements in an array are uninitialized.
- For initialized arrays, you define the number of elements in each dimension and the contents of each element.
- Uninitialized arrays have dimension but no elements.

Slide 7

Demo: **Viewing Arrays** 10 min

Location: n/a – Start with a new VI.
Show how to create a new array from the Controls palette and from a block diagram terminal or wire.

Creating array controls in LabVIEW:

- You can place any data type in an array shell except an array.
- You cannot have an array of arrays; use a 2D array instead.
- Emphasize that this is a two-step process. Remind them they must place a data type inside the array shell.

Demonstrate the following on your computer:

- Create a numeric array.
- Point out index and data object components.
- Show how to create a 2D array.
- Show how to display multiple array elements.
- Show that index elements always reference the upper-leftmost object in the array display.
- Show how elements in an array are initially grayed out, indicating that a portion of the array has not been defined.

Demo: Creating an Array Constant

- For a new array:
  - Select Array Constant from the Functions palette on the block diagram.
  - Place a constant, such as a numeric,into the array shell.
  - Add a second dimension, if necessary, by resizing the index.
- From a block diagram terminal or wire:
  - Right-click and select Create»Constant.
- Note again that the data types are grayed-out and must be manually defined if the user wants to store values in the array constant.

Slide 8

B. Common Array Functions

Objective: Create and manipulate arrays using built in array functions.

Slide 9

Multimedia: *Common Array Functions* — 10 min

Location: <Exercises>\LabVIEW Core 1\Multimedia\Common Array Functions

Use LabVIEW's built-in array functions to create and manipulate arrays.
- Array Palette
- Array Size
- Array Subset
- Build Array
- Index Array

Transition: Let's look at some more array functions.

Slide 10

Initialize Array
- Use this function if you want to create arrays to which every element in the array is initialized to the same value.
- Is a resizable node

Slide 11

Insert Into Array
Use this function to insert an element or subarray at the index specified.

Slide 12

Delete From Array
- Use this function to delete elements from the array.
- Explain the length and the index input.

Slide 13

Array Max and Min
Highlight that this function outputs the maximum and minimum value in the array and their respective indices.

Slide 14

Search 1D Array Function
- Searches for an element in a 1D array starting at the index value specified
- Highlight that this function returns a -1 if the element is not there in the 1D array.

Slide 15

Activity 5-1: Using Array Functions — 15 min
Ask students to complete table 5-1 in the participant guide.

Slide 16

Activity 5-1: Using Array Functions Answers
Discuss the answers with the students.

Slide 17

Activity 5-1: Using Array Functions Answers
Discuss the answers with the students.

Slide 18

C. Polymorphism

Objective: Understand the ability of various VIs to accept input data of different data types.

Slide 19

Polymorphism

- Highlight that functions are polymorphic to varying degrees—none, some, or all of their inputs can be polymorphic.
- Some function inputs accept numeric values or Boolean values. Some accept numeric values or strings.
- Some accept not only scalar numeric values but also arrays of numeric values, clusters of numeric values, arrays of clusters of numeric values, and so on.

Slide 20

Arithmetic Functions Are Polymorphic

- LabVIEW arithmetic functions are polymorphic.
  - Inputs to arithmetic functions can be of different data types.
  - The function automatically performs the appropriate operation on unlike data types.
  - LabVIEW arithmetic functions greatly simplify array arithmetic.
- Examples of polymorphism on this slide:
  - Scalar+Scalar: Scalar addition.
  - Scalar+Array: The scalar is added to each element of array.
  - Array+Array: Each element of one array is added to the corresponding element of other array.

In case students ask, polymorphism does not perform matrix arithmetic when inputs are 2D arrays. (For example, multiplying two 2D array inputs with the Multiply function performs an element by element multiplication, not matrix multiplication).

Slide 21

D. Auto-Indexing

Objective: Use auto-indexing inputs and outputs to create graphs and arrays.

Slide 22

Auto-Indexing

- For Loops and While Loops can index and accumulate arrays at their boundaries. This is known as auto-indexing.
- The indexing point on the boundary is called a tunnel.
- Is the default behavior for For Loops.
- The While Loop default is auto-indexing disabled. Only one value (the last iteration) is passed out of the While Loop by default.
- Is enabled/disabled by right-clicking on a tunnel.
- Enable auto-indexing to collect values within the loop and build the array. All values are placed in the array upon exiting the loop.
- Disable auto-indexing if you are interested only in the final value.
- Produces arrays that are always equal in size to the number of iterations of the loop.

Slide 23

Waveform Graph

- Is a graphical display of data.
- Displays one or more plots of evenly sampled measurements.
- Highlight that it is used to plot pre-generated arrays of data and can display plots with any number of data points.

Slide 24

Charts vs. Graphs

Highlight the difference between charts and graphs

Slide 25

Auto-Indexing with a Conditional Tunnel

- You can determine what values LabVIEW writes to the loop output tunnel based on a condition you specify by right-clicking the loop output tunnel and selecting Tunnel Mode»Conditional from the shortcut menu.
- Because of the conditional tunnel, the Values less than 5 array contains only the elements 2, 0, 3, and 1 after this loop completes all iterations.
- Mention the concatenating tunnel mode too.

Slide 26

Creating Two-Dimensional Arrays

- Explain how the Auto-indexing output creates a 2D array
- Explain the different line thicknesses in the wire connecting the Random Number function to the 2D Array Indicator.

Slide 27

Auto-Indexing Input

- Explain that a For loop iterates the smallest number of times when there are arrays of different sizes wired to the For Loop.
- Highlight that calculations are performed on each element in the array
- Do not need to wire the N terminal. For Loop iterates depending on the number of elements in the array.

Slide 28	**Auto-Indexing Input—Different Array Sizes** • The For Loop executes the number of times equal to the number of elements in the array. • The Run button is not broken.	
Slide 29	**Exercise 5-1: Manipulating Arrays** **Goal**: Explore arrays using various LabVIEW functions. Open the Participant Guide and review the scenario with students.	45 min
Slide 30	**Discussion: Exercise 5-1: Manipulating Arrays** **Question:** In the All Data Channel case, how can we verify that the two approaches yield the same results? **Answer**: You can use the comparison and Boolean functions to compare if the All Data Channel array is the same. Comparison functions are also polymorphic. Use the Equal? function followed by the And Array Elements function. Typically, you can't perform a direct comparison on double-precision values. You would need to compare the values within some acceptable range. However, in this case, the polymorphic Add operation should be an exact equivalent to the Add function in an auto-indexed For loop.	
Slide 31	**E. Clusters** **Objective**: Identify when to use clusters and be able to create them.	
Slide 32	**Clusters** Recap from getting started.	
Slide 33	**Clusters vs. Arrays** Highlight the difference between clusters and arrays	

Slide 34 **Demo: Create a Cluster Control and Constant** 20 min

Location: n/a – Start with a new VI.
Creating Cluster Controls

- For a new cluster:
    - On the front panel, select Cluster from the Controls palette.
    - Place a data object into the cluster shell.
    - Place additional data objects, if necessary, into the shell.
- From block diagram terminal or wire:
    - Right-click and select Create»Control or Create»Indicator.

Creating Cluster Constants

- For a new cluster:
    - On the block diagram, select Cluster Constant from the Functions palette.
    - Place a constant into the cluster shell.
    - Place additional data objects, if necessary, into the cluster shell.
- From block diagram terminal or wire:
    - Right-click and select **Create»Constant**.

Slide 35 **Cluster Order**

- Highlight the importance of why this is important while creating a SubVI
- View and modify the cluster order by right-clicking the cluster border and selecting Reorder Controls In Cluster.

Slide 36 **Autosizing Clusters**

- Highlight how this is a design element and that this helps keep the cluster organized
- NI recommends the following:
    - Arrange cluster elements vertically.
    - Arrange elements compactly.
    - Arranges elements in their preferred order.

Slide 37 **Disassembling a Cluster**

- Highlight when to use the following functions:
- Unbundle by name
- Unbundle

Slide 38 **Modifying a Cluster**

- Bundle by Name
- Bundle

Slide 39	**Demo: Creating a Cluster on the Block Diagram** **Location:** n/a – Start with a new VI. • Use the Bundle function to programmatically create a cluster on a block diagram. • If the elements that are bundled have labels, you can access them using the Unbundle By Name function. Otherwise use the Unbundle function. This technique is typically used to create multi-plot charts which is discussed in more detail later.	15 min
Slide 40	**Multi-plot Graphs/Charts and XY Graph** • Highlight that the Bundle function is often used to create multi-chart plot charts and XY plots. • The Build Array function is used to create multi-plot waveform graphs.	
Slide 41	**Plotting Data** • Use the Context Help and shipping examples to investigate all the possible ways to assemble data for various graphical displays. • LabVIEW examples and help provide extensive information on wiring data to the various graphical displays.	
Slide 42	**Error Clusters** • LabVIEW uses error clusters to pass error information • An error cluster contains the following elements: ▪ **status**—Boolean value that reports True if an error occurs. ▪ **code**—32-bit signed integer that identifies the error. ▪ **source**—String that identifies where the error occurred..	
Slide 43	**Exercise 5-2: Clusters** **Goal**: Create clusters, reorder clusters, and use the cluster functions to assemble and disassemble elements. Open the Participant Guide and review the scenario with the students.	30 min
Slide 44	**Discussion: Exercise 5-2: Clusters** **Question**: What would happen if you reordered cluster elements? **Answer**: Both the subVI and calling VI would continue to work as expected since the only cluster functions used are Unbundle By Name and Bundle By Name. A change to element order in a cluster does not break these functions. **Question**: What would happen if you added a cluster element to Weather Data Out? **Answer**: Both the subVI and the calling VI would be broken because of mismatching cluster data types.	

Slide 45

F. Type Definitions

Objective: Identify and determine when to use a type definition, strict type definition, or control.

Slide 46

Control Options

Discuss the 3 control options.

Type definition

- A type definition is a master copy of a custom data type (control, indicator, or constant).
    - A custom data type is saved in a .ctl file.
    - Instances of a type def are linked to the .ctl file.
- Instances can be controls, indicators, or constants.
- When the type def changes, instances automatically update.
    - Changes include data type changes, elements added, elements removed, and adding items to an enum.

Control

- Instances are not linked to a .ctl file.
- Each instance is an independent copy of the control.
- Used to create controls that behave like existing controls but look different.

Strict Type Definition

- Strict type definitions are similar to a type definition in that:
    - All instances link to .ctl file.
    - When attributes or data types change, all instances update.
    - Examples: Changing a knob to a dial, a round LED to a square LED, or a double to an integer.
- Strict type definitions enforce every aspect of a instance except label, description, and default value.
- Use strict type definitions to ensure all instances of a control have the same appearance on the front panel.

Slide 47

Demo: **Difference between Control, Type def, and Strict Type Def**

10 min

Location: <Exercises>\Demonstrations\TypeDef Demo\TypeDef Demo.lvproj

1. Open `Custom Control.ctl`, `TypeDef.ctl`, and `Strict TypeDef.ctl` and arrange them vertically on the left side of your screen.
   a. Point out the dropdown menu that indicates the type of CTL.
2. Open `TypeDef Demo.vi` and show its front panel next to the CTLs.
3. Add a Boolean push button to the cluster in each CTL.
   a. In `TypeDef Demo.vi`, point out that the **TypeDef** and **Strict TypeDef** controls are grayed out and Custom Control did not change.
4. For each CTL, select **File»Apply Changes**.
   a. Point out that this option is not available for the custom control. Each instance of the custom control is independent of the source CTL.
   b. In `TypeDef Demo.vi`, point out that **TypeDef** and **Strict TypeDef** updated to reflect the data changes to the source CTLs.
5. Change the background color of each CTL cluster.
   a. In `TypeDef Demo.vi`, point out that the **TypeDef** and **Strict TypeDef** controls are grayed out because the VI detects that the source CTLs have changed, but does not know the extent yet.
6. For each CTL select **File»Apply Changes**.
   a. In `TypeDef Demo.vi`, point out that **Strict TypeDef** updated to reflect the cosmetic change to the source CTL.
7. In `TypeDef Demo.vi`, attempt to add a new control to each cluster.
   a. Only **Custom Control** allows you to make datatype changes.
   b. The other clusters are tied to the data type of the source CTL.
8. In TypeDef Demo.vi attempt to modify the background color of each cluster.
   a. **Custom Control** and **TypeDef** allow you to make this cosmetic modification to instances of the CTLs.
   b. **Strict TypeDef** does not allow you to make even cosmetic changes to the appearance of the cluster.

Slide 48

Demo: Creating and Indentifying Type Definitions

5 min

Location: <Exercises>\Demonstrations\TypeDef Demo\TypeDef Demo.lvproj

Creating Type Definitions
- Open TypeDef Demo.vi.
- Right-click the Custom Control cluster and select Make Type Def.
- Right-click the cluster again and select Open Type Def.
- Edit the control (rearrange contents, change colors, etc).
- Save control as a .ctl file.

Identifying Type-Definitions
- Look for a glyph marking the upper left corner of terminals and constants.
- Hover cursor over glyph to view tip strip.
- View Context Help while hovering cursor over terminal or constant

---

Slide 49

Exercise 5-3: Type Definition

30 min

Goal: Create and modify a type-defined cluster control and use the type definition in a calling VI and a subVI.

Open the Participant Guide and go over the scenario with the students.

Slide 50

Discussion: Exercise 5-3: Type Definition

Question: Now that the Weather Data control is saved as a type definition, how many instances of the cluster would need to be updated to add wind speed data?

Answer: One. You add elements to the type definition. Apply and save changes.

Converting from a cluster to a type definition requires a one-time update to each cluster. However, once you have your data in a type definition, adding or removing elements is a simple update to the type definition control.

Slide 51-59

Activity 5-2: Lesson Review

20 min

Refer students to the Participant Guide to answer the questions before showing the slides and discussing the answers with the group.

Lesson 6. Using Decision-Making Structures

Objective: Learn to create different decision-making structures and be able to identify applications where using these structures can be beneficial.

This lesson contains the following topics:

A. Case Structures
B. Event-Driven Programming

Duration: 2 hours 15 min

	Content/Narrative	**Duration**
Slide 1	State the lesson objective. Briefly outline the topics covered in this lesson.	
Slide 2	**A. Case Structures** **Objective:** The learner will be able to recognize and use the basic features and functionality of Case Structures.	
Slide 3	**Activity 6-1: Case Structure Review** Review the concept and components of the Case Structure. Give the students time to answer all four questions before discussing as a class. For students who did NOT complete the Getting Started module beforehand, they can refer to the Case Structures Overview content following the activity. **Question**: What is the purpose of the Case Structure? **Answer:** A Case Structure executes one of its subdiagrams based on an input value. Each Case Structure will have two or more subdiagrams or cases. Case Structures are similar to case statements or if...then...else statements in text-based programming languages. **Question**: How many cases does a Case Structure execute at a time? **Answer:** One. **Question**: What is the purpose of the Case Selector Label? A**nswer**: Show the name of the current case and enable you to navigate through different cases. **Question**: What is the purpose of the Selector Terminal? **Answer**: Lets you wire an input value to determine which case executes. You can wire a variety of data types to the Selector Terminal:	5 min

Slide 4 **Demo: Creating and Configuring Case Structures** 15 min

Location: n/a – Start with a new VI.

1. Open a blank VI and create a Case Structure
2. Add a numeric constant to one case.
3. Right-click on the edge of the Case Structure and describe the different things that you can do from the shortcut menu
   A. Remove or replace the structure.
   B. Add, duplicate, remove, or rearrange cases (if there are more than two).
   C. Switch cases.

The shortcut menu options will vary some depending on the data type that you wire to the Selector Terminal.

Transition: "Now that we understand how to configure a Case Structure, let's discuss the different types of values that you can wire to the Selector Terminal."

Slide 5 **Selector Terminal Data Types**

- **Boolean**: Creates two cases: True and False
- **Integer or string** - The Case Structure can have any number of cases.
  - Developer must add a case for each integer or string as necessary.
  - Include a Default case to avoid having to include every possible input.
  - For integers, the representation that you choose will determine the range of possible values.
- **Enums -** Gives users a list of items from which to select a case.
  - The Case Structure can be configured to include a case for each item in the enumerated type control.
- **Error Cluster** - Creates two cases: Error and No Error
  - Use this option to execute the code if there is no error and skip the code if there is an error.

Slide 6 **Input and Output Tunnels**

- Like other structures, you can pass data into and out of Case Structures by using tunnels.
- Point out the visual differences between:
  - Tunnels that have been wired for all cases
  - Tunnels that are unwired will cause a broken Run arrow
  - Tunnels marked as Use Default If Unwired.
- Default values for tunnels:
  - Numeric: 0
  - Boolean: FALSE

Slide 7 **Demo: Selector Terminal Types and Tunnels** 10 min

Location : <Exercises>\Demonstrations\Case Structures\Case Structures.vi

A pre-built VI with various selector terminal is available in the following demonstration folder:

- Show how wiring different data types to the selector terminal changes the case selector label.
- "You should always wire the selector terminal BEFORE trying to modify the case selector label."
- As you work through these different terminal types, show how the Shortcut menu changes.
    - Boolean -A newly-created Case structure defaults to a Boolean input
        - Integer - If an undefined integer or strings is wired to the terminal, LabVIEW uses the Default case.
        - "Include a Default diagram to avoid listing every possible input value."
        - Show that you can specify a range of values.
        - Shortcut menu includes Default options and Radix.
    - String
        - Like integers, strings can have any number of values, so you should include a Default case.
        - By default, string values are case sensitive.
        - Shortcut menu includes option for **Case Insensitive Match** for the string text.
    - Enums - Show how to add a case for every value of the Enum
    - Error Cluster - Explain how Error codes only execute if an error occurs.
- Demonstrate other functionality of the Shortcut Menu
- Show how the **Run** arrow is broken if a tunnel is not wired in each case.
    - Configure the tunnel to Use Default if Unwired.
    - "Be careful about using the Use Default If Unwired option on Case structure tunnels for the following reasons:
        - Adds a level of complexity to the code.
        - Complicates debugging your code."
    - Wire a value to the tunnel - Point out the change in the appearance of the tunnel.

Slide 8 **Exercise 6-1: Temperature Warnings VI—With Error Handling** 25 mins

Goal: Modify the Temperature Warnings VI from the Data Structures lesson to include a Case Structure that will compare the maximum and minimum values and only execute the code if the maximum is greater than the minimum.

If the student's Weather Warnings.lvproj in the <Exercises>\ LabVIEW Core 1\Weather Warnings folder is not complete, the student can copy the solution from <Solutions>\LabVIEW Core 1\Exercise 5-3 into their working <Exercises> directory.

Slide 9

Discussion: Exercise 6-1: Temperature Warnings VI—With Error Handling

Open the block diagram for the solution to the Exercise.

Question: "On step 6, when you predicted the values of **Warning Text** and **Warning?** Did your values match the values specified in the third testing step? If not, do you have any questions about why?"

Question: What happens if all the values are 10? How could you fix this?

Answer: You get a freeze warning. There are multiple ways you can fix this.

- One possibility is to not allow Min Temperature and Max Temperature to be equal
- Replace the **Greater** function with a **Greater or Equal** function.
- Change the string constant in the True case to "Upper Limit <= Lower Limit."

Question: Are all output tunnels defined? What happens if an output is not defined?

Answer: All output tunnels are defined for this exercise. If an output isn't wired, the Run arrow would be broken.

To use the Use Default if Unwired option, one would need to enable it through a shortcut menu selection.

Slide 10

B. Event-Driven Programming

Objective: Recognize basic features and functionality of event structures.

Slide 11

Demo: **Event-Driven Scenario**

10 mins

Location: <Exercises>/Demonstrations/Event-Driven Programming/ Event-Driven Programming.lvproj

1. Open Event-Driven Scenario (Polling Method).vi. Only show the front panel. Do NOT show the block diagram yet.
2. Run Event-Driven Scenario (Polling Method).vi
3. Click the Time Check, Acquire Data, and Temperature (deg C) controls a few times to demonstrate that these **front panel actions trigger block diagram code to execute** (e.g. get current time, acquire waveform data, convert temperature unit).

- **Question:** How would you implement this functionality in a VI?
    - o Encourage students to suggest potential approaches.
    - o Based on what the students have learned so far, they would only know how to implement this functionality by constantly polling the front panel controls.

4. Show the block diagram to the students, and explain the polling method of implementing this functionality.
    - o While Loop continuously polls the value of the controls at the rate set by the Wait Until Next ms Multiple function and executes the corresponding code in the Case structures.

Slide 12

Polling versus Events

•

Emphasize the benefits of using events over polling for the example scenarios discussed in this section.

Polling

- Continuous code check
- CPU intensive
- Slower to respond and can fail to detect changes

Events

- On-demand
- Reduce CPU work
- Handles all events in the order they occur

Transition: There is a better method for the front panel event to trigger executing code on the block diagram written to handle that event.

Slide 13 **Multimedia**: Event-Driven Programming 15 min

Location: <Exercises>\LabVIEW Core 1\Multimedia\ Event-Driven Programming

Students will be able to do the following after viewing the multimedia module:

- Describe how Event Structures implement event-driven programming
- Describe the parts of an Event structure: event selector label, timeout terminal, event data node, event filter node)
- Recall that Event structures are usually placed inside a While Loop
- NOTE: Students will NOT know anything about the Event structure configuration window.

Transition:
In the Event-Driven Programming.lvproj, open the Event-driven Scenario (Event Method).vi and show the same function is implemented with an Event structure.

Now you know Event structures implement event-driven programming. Next, we will talk about how to configure the Event structure.

Slide 14 **Configuring the Event Structure**
Quickly describe how to configure the Event Structure and notify/filter events. You will show students in the next demo.

Slide 15 **Edit Events Dialog Box**
Go through slide quickly as an overview because you will show students in the next demo.

Slide 16 **Notify and Filter Events**
Focus on explaining the difference between Notify Events and Filter Events.

- Notify Events (green arrow)
  - User action has already occurred and LabVIEW has processed the event.
- Filter Events (red arrow)
  - User action has already occurred and LabVIEW has NOT processed the event.
  - Filter events allow you to override default behavior for event.

Slide 17 **Demo**: **Configure and Use Events** 10 min

Location: n/a – Start with a new VI.

- For the demonstration: Drop one or two controls on front panel, including a Stop button. On the block diagram, drop an Event structure in a While Loop. Configure the Event structure to respond to the front panel controls and then terminate using the Stop value change.
- Describe the event registration as part of the demonstration.
    - Event Registration:
    - Collection of steps that happen behind the scenes to prepare the LabVIEW Event Handler to handle the event.
    - Event registration involves:
    - Allocation of memory to queue up events.
    - Spawning of pieces of code that run behind the scenes to catch events that happen. These pieces of code are called listeners or observers.
- Key points:
    - LabVIEW registers events that you configure with the Edit Events dialog box.
    - When a registered event occurs, LabVIEW queues the event until the Event Handler executes.
    - It is not possible to miss events or process them out of order.

Events are unregistered when the VI becomes idle.

Slide 18 **Exercise 6-2: Converting a Polling Design to an Event Structure Design** 45 mins

Goal: Explore a polling-based application and then convert it to an event-based application and compare the difference in performance.

Students gain experience using and configuring the Event structure including Value Change event, Mouse Down event, and Panel Close? filter event.

Slide 19 **Discussion: Exercise 6-2: Converting a Polling Design to an Event Structure Design**

Question: How many times did the loop run with polling?

Answer: The While Loop has no Wait function, and it will keep executing as fast as the CPU will let it. As a result, the While Loop will run many times.

Question: How many times did the loop run when you changed the VI to use an Event structure?

Answer: The While Loop will only run one iteration per event.

Slide 20 **Caveats and Recommendations**

For a complete list of Caveats and Recommendations, refer to LabVIEW Help topic: *Caveats and Recommendations when Using Events in LabVIEW*.

Slide 21	**Think about the VIs that you will need to develop at your job.** **Group discussion**: How would you apply event-driven programming to your own applications at work? Encourage students to talk through specific examples of VIs that they might need to create at work that would use event-based programming. It's also possible that students might not need to create any VIs that require using event-based programming. Encourage students to give examples that justify not using event-based programming. For example, a simple VI that just runs, returns results, and immediate stops running does not need to use event-based programming because the user does not interact with the front panel while the VI is running.	
Slide 22-28	**Activity 6-2: Lesson Review** Refer students to the Participant Guide to answer the questions before showing the slides and discussing the answers with the group	5 min

Lesson 7. Modularity

Objective: Recognize the benefits of reusing code and create a subVI with a properly configured connector pane, meaningful icon, documentation, and error handling.

This lesson contains the following topics:

A. Understanding Modularity
B. Icon
C. Connector Pane
D. Documenting Code
E. Using SubVIs

Duration: 2 hours 10 min

	Content/Narrative	**Duration**
Slide 1	State the lesson objective. Briefly outline the topics covered in this lesson.	
Slide 2	**A. Understanding Modularity** **Objective:** Recognize the benefit of using modular code and identify sections of code that could be reused.	
Slide 3	**Modularity and SubVIs** LabVIEW uses subVIs to create modularity SubVIs: • Similar to subroutines (or functions) in text-based programming • Icon and connector pane displayed in upper right corner • Icon used to identify the VI when used as a subVI on block diagram	
Slide 4	**SubVIs—Reusing Code** Repeated code can become a SubVI	
Slide 5	**SubVIs** • Every VI has icon and connector pane. • Icon and connector pane correspond to the function prototype in text-based programming.	

Slide 6	**B. Icon** **Objective**: Recognize the characteristics of a good icon and use the LabVIEW Icon Editor to create a custom icon.	
Slide 7	**Purpose of Icon** Graphical representation of a VI identifies the subVI on the block diagram of a calling VI	
Slide 8	**Characteristics of a Good Icon** Convey the functionality of the VI using relevant graphics and descriptive text.	
Slide 9	**Creating Icons—Icon Editor** Customize the icon so you can identify the VI when used as a subVI	
Slide 10	**Demo: Creating an Icon** **Location:** n/a – Start with a new VI. Open a blank VI and demonstrate how to create an icon. Show each of the tabs and cover the following high points: • Icon text and using colors • Searching for glyphs • Use templates to create a common look for related VIs, Example: DAQmx VIs.	5 min
Slide 11	**C. Connector Pane** **Objective:** Select and configure a connector pane for a SubVI.	
Slide 12	**Patterns** • Connector pane is displayed next to icon • Many patterns available depending on how many inputs and outputs required • Point out that the 4:2:2:4 is the default pattern.	
Slide 13	**Assigning Terminals** Explain and maybe do a quick demo how to click connector pane terminal and then click the control or indicator to assign the terminal	
Slide 14	**Standards** • Default connector pane • Inputs on the left, outputs on the right • References on the top • Error clusters on the bottom	

Slide 15

D. Documentation

Objective: Explain how to document code in LabVIEW using descriptions and tip strips, and 4 methods for documenting code on the block diagram.

Slide 16

Creating Descriptions and Tip Strips

- Description appears in Context Help Window
- Tip strip displays when cursor is over object while VI runs

Quick Demo

Starting VI: Temperature Warnings.vi

Create documentation for a control or indicator on the front panel and then demonstrate how is appears in Context Help.

Show Documentation in the Properties dialog box for a VI and then place it as a subVI on a blank VI and show the documentation in Context Help.

Slide 17

Documenting Block Diagram Code

- Use comments to document algorithms and add reference information.
- Label structures to specify the main functionality.
- Label long wires to identify their use/contents.
- Label constants to specify the nature of the constant.
- It is not always necessary to show labels on functions and subVIs if they make the block diagram cluttered. A developer can find information about a function or subVI by using the Context Help window.

Slide 18

E. Using SubVIs

Objective: Demonstrate how to place subVIs on the block diagram, explain terminal settings and error handling, and create subVIs from a section of existing code.

Slide 19

Placing SubVIs on the Block Diagram

- Drag the VI from the Project Explorer to the block diagram.
- Click Select a VI on the Functions palette and then navigate to the VI.
- Drag the icon from an open VI to the block diagram of another VI.
- Use Quick Drop to search an open project for VIs contained in the project.

Slide 20	**Terminal Settings** You can designate which inputs and outputs are required, recommended, and optional to prevent users from forgetting to wire subVI terminals. • Required means that the block diagram on which you place the subVI will be broken if you do not wire those inputs. • View the Context Help for the subVI to help identify the terminal settings. • Describe how to change a terminal requirement. • LabVIEW sets inputs and outputs to **Recommended** by default. • Set a terminal setting to required only if the VI must have the input or output to run properly.	
Slide 21	**Handling Errors** Use Case structures to handle errors passed into the VI	
Slide 22	**Handling Errors** Avoid using LabVIEW error handler VIs inside SubVIs	
Slide 23	**Convert a Section of a VI to a SubVI** • Select the section of code to reuse • Select **Edit»Create SubVI**	
Slide 24	**Exercise 7-1: Create an Icon and Connector Pane** **Goal:** Create the icon and connector pane for a VI so you use the VI as a subVI. Call the subVI from a test VI.	50 min
Slide 25	**Discussion: Exercise 7-1: Create an Icon and Connector Pane** **Question:** Do the terminal names in the calling VI need to match the subVI terminal names? **Answer:** No. **Question:** Do the data types in the calling VI need to match the subVI terminal data types? **Answer:** Yes. For example, a string cannot be wired directly to a numeric and vice versa. However, LabVIEW can coerce the numeric data type to the subVI data type (such as coercing an integer to a double). In addition, you can program the subVI to support polymorphic inputs.	
Slide 26-30	**Activity 7-1: Lesson Review** Refer students to the Participant Guide to answer the questions before showing the slides and discussing the answers with the group	5 min

Lesson 8. Acquiring Measurements from Hardware

Objective: Understand the difference between DAQ systems and instrument control and how LabVIEW connects to hardware to get real-world measurements.

This lesson contains the following topics:

A. Measuring Fundamentals with NI DAQ Hardware
B. Automating Non-NI Instruments

Duration: 1 hour 40 min

	Content/Narrative	**Duration**
Slide 1	State the lesson objective. Briefly outline the topics covered in this lesson.	
Slide 2	**A. Measuring Fundamentals with NI DAQ Hardware** **Objective**: Recognize the components of a DAQ system and practice connecting to hardware.	
Slide 3	**DAQ—Measuring physical phenomenon with a computer.** Highlight LabVIEW's ability to measure signals with different NI DAQ devices and different sensors. Have the students describe their experience with data acquisition.	
Slide 4	**Multimedia**: *Measurement Fundamentals with NI DAQ* **Location**: <Exercises>\LabVIEW Core 1\Multimedia\ Measurement Fundamentals with NI DAQ In the multimedia module, students learn how to connect to DAQ hardware using NI MAX and LabVIEW and how to set up a basic DAQmx application. **Transition:** Tell students they'll have a chance to practice connecting to hardware in the next two exercises.	10 min
Slide 5	**Exercise 8-1: Use NI MAX to Examine a DAQ Device** **Goal**: Use NI MAX to connect to either the BNC-2120 or a simulated device. Open the Participant Guide to the exercise let students know what to do if they don't have hardware.	20 min
Slide 6	**Discussion**: **Exercise 8-1: Use NI MAX to Examine a DAQ Device** Ask students if they have any questions about NI MAX. **Transition:** Tell students they'll have a chance to practice connecting to hardware from the LabVIEW block diagram in the next exercise.	

Slide 7	**3 Ways to Connect with a DAQ Device** Recap the 3 ways to connect with a DAQ device, which were introduced in the multimedia module, Measuring Fundamentals with NI DAQ.	
Slide 8	**Exercise 8-2: Programming with the DAQmx API** **Goal**: Explore a DAQmx example program and modify the VI to use a digital trigger. Open the Participant Guide to the exercise and go over the scenario.	30 min
Slide 9	**Discussion: Exercise 8-2: Programming with the DAQmx API** **Question**: What types of VIs did you use outside of the While Loop? **Answer**: VIs that only need to be called once (not continuously) such as DAQmx Create Virtual Channel and DAQmx Clear Task. **Transition:** LabVIEW can also communicate with benchtop instruments like oscilloscopes and amplifiers	
Slide 10	**B. Automating Non-NI Instruments** **Objective**: Recognize the components of an instrument control system and practice connecting to hardware.	
Slide 11	**LabVIEW controls instruments through different buses.** Highlight LabVIEW's ability to automate and control benchtop instruments through different buses (different cable connectors indicate different buses). Have the students describe their experience with benchtop instruments, such as oscilloscopes or analyzers. **Transition**: The following multimedia module goes into more detail on instrument control and buses.	
Slide 12	**Multimedia**: *Automating Non-NI Instruments* **Location**: <Exercises>\LabVIEW Core 1\Multimedia\ Automating Non-NI Instruments In this module, students will recognize the relationship between LabVIEW and instrument control and how to use VISA to communicate with third-party instruments. **Transition:** Tell students they'll have a chance to practice connecting to hardware in the next two exercises.	10 min

Slide 13

Exercise 8-3: Instrument Configuration with NI MAX — 15 min

Goal: Configure the NI Instrument Simulator and use MAX to examine the GPIB interface settings, detect instruments, and communicate with an instrument

Open the Participant Guide to the exercise and let the students know that they must first set up the NI Instrument Simulator before they can use MAX to connect with and explore the device.

Slide 14

Discussion: Exercise 8-3: Instrument Configuration with NI MAX

Ask students if they have any questions about NI MAX and instrument communication.

Slide 15

Simplify Instrument Control.

Highlight the benefits of using instrument drivers over creating LabVIEW applications with VISA or other bus protocols.

- Is a high-level API (VIs perform multiple instructions).
- Does not require knowledge of different bus protocols
- Does not require learning low-level programming commands for each instrument.

Transition: The next slide zooms in on an example instrument driver application.

Slide 16

Instrument driver VIs for an Agilent digital multimeter.

Highlight the role of the instrument driver VIs in the block diagram.

- Initializes the Agilent 34401 digital multimeter (DMM)
- Uses a configuration VI to choose the resolution and range
- Selects the function
- Uses a data VI to read a single measurement
- Closes the instrument

Slide 17

Exercise 8-4: Exploring Instrument Drivers — 10 min

Goal: Install an instrument driver for the NI Instrument Simulator and explore the example programs that accompany it.

Open the Participant Guide to the exercise and let the students know that they can either download the instrument driver from ni.com or install it from disk.

Slide 18

Discussion: Exercise 8-4: Exploring Instrument Drivers

Question: How would modify the example programs if you wanted to continuously acquire data?

Answer: Place a While Loop around the Read function.

Slides 19-25	**Activity 8-1: Lesson Review** Refer students to the Participant Guide to answer the questions before showing the slides and discussing the answers with the group.	5 min

Lesson 9. Accessing Files in LabVIEW

Objective: Describe the basic concept of file I/O and apply the appropriate File I/O functions to a given scenario.

This lesson contains the following topics:

A. Accessing Files from LabVIEW
B. High-Level and Low-Level File I/O
C. Accessing Other Resources

Duration: 1 hour

	Content/Narrative	**Duration**
Slide 1	State the lesson objective. Briefly outline the topics covered in this lesson.	
Slide 2 ▤	**A. Accessing Files from LabVIEW** **Objective**: Identify the steps for writing and reading files from a LabVIEW application.	
Slide 3	**Typical File I/O Operations.** File I/O operations pass data to and from files. Have the students describe other examples of using the Open/Read-Write/Close pattern. Examples: Programming hardware with DAQmx, editing a text document in a word processor	
Slide 4	**File I/O Palette for File Operation Functions** Introduce the File I/O palette. The File I/O palette contains functions for generic and specific file types.	

Slide 5

B. High-Level and Low-Level File I/O Functions

Objective: Identify when to use high-level and low-level File I/O functions.

Slide 6

High-level File I/O
High-level File I/O performs open, read/write, and close operations.

Mention pros/cons of using high-level file I/O functions.

- **Pros**: Simpler block diagram.
- **Cons**: Might not be as efficient as individually configured functions.

Briefly describe the differences between the high-level file I/O functions:

Function	Type of Data	File Formats
Write/Read to Spreadsheet file	1D or 2D arrays	Text
Write/Read to Measurement file	Signal (dynamic data type)	.lvm (text), .tdms (binary)
Waveform file	Waveform data	Text

Slide 7

Exercise 9-1: Exploring High-Level File I/O 20 min
Goal: Run the Spreadsheet Example VI, explore the block diagram, and view the 2D array text file that is created.

Open the Participant Guide to the exercise and go over the Scenario section describing the tasks that the Spreadsheet Example VI completes.

Slide 8

Discussion: **Exercise 9-1: Exploring High-Level File I/O**

Question: How would you include more columns of data in the log file?

Answer: Add another input to the Build Array function. The data that is wired to that input will be transposed into a new column.

Transition: High-level file I/O functions can work well for complete data sets but when it comes to data streaming, low-level file I/O functions are more efficient.

Slide 9

Low-level File I/O
Point out that compared to high-level file I/O, low-level file I/O provides individual functions for each step.

Slide 10	**File Refnums** Refnums are: • Temporary pointers to a resource, such as a file, device, or object. • Unique identifiers. Highlight the **file path** input details for the Open/Create/Replace File function: • You can use a control (file dialog) or constant. • The **file path** input is the absolute path to the file. • If **file path** is not wired, the function displays a dialog box to select a file. • If you specify an empty or relative path, this function returns an error. • Note that you do not have to handle the file reference when you work with the high-level file I/O functions.	
Slide 11	**Streaming Data to Disk** Low-level File I/O VIs are more efficient when writing to a file in a loop. **Question**: How would the VI behave differently if you used the Write To Spreadsheet VI in place of the low-level file I/O VIs? **Answer**: The VI would open and close the file with every iteration of the loop, slowing the writing of data to the file.	
Slide 12	**String Functions.** The Write to Text File VI and Write to Spreadsheet VI require strings. • Convert numeric and Boolean data types to a string data type. • Use String constants like Tab and Line Feed to format text. • Use escape codes in the **format string** input of the Format Into String function to insert non-displayable characters.	
Slide 13	**Exercise 9-2: Temperature Monitor VI – Logging Data** **Goal**: Modify a VI to create an ASCII file using disk streaming. Open the Participant Guide to the exercise and go over the scenario.	30 min
Slide 14	**Discussion: Exercise 9-2: Temperature Monitor VI – Logging Data** **Question**: What would happen if you used the Write to Measurement File Express VI inside the While loop? **Answer**: The VI would run slower and be more inefficient. Because the Write to Measurement File Express VI is a high-level File I/O VI, the While Loop would open, write, and close the file every iteration of the While Loop.	

Slide 15

C. Comparing File Formats

Objective: Recognize different options for logging data to disk.

Slide 16

Common Log File Formats
Briefly describe the different file formats.

- ASCII—Human-readable text file where data is represented as strings. Commonly used for low-speed DAQ.
- LVM—Format built on ASCII, the LabVIEW measurement data file (.lvm) is a tab-delimited text file.
- Binary—Not human-readable. Commonly used for high-speed and multi-channel DAQ.
- TDMS—An NI-specific binary file format that contains data and stores properties about the data.

Tell students they learn more about File I/O techniques and different formats in Core 2.

Slide 17

Comparing File Formats
Highlight the file format differences in the table.

Use ASCII files in the following situations:

- You want to access the file from another application.
- Disk space and file I/O speed are not crucial.
- You do not need to perform random access reads or writes.
- Numeric precision is not important.

Slides 18-22

Activity 9:1: Lesson Review 3 min
Refer students to the Participant Guide to answer the questions before showing the slides and discussing the answers with the group

Lesson 10. Using Sequential and State Machine Programming

Objective: Recognize the benefits of sequential and state-based algorithms and apply techniques in LabVIEW to enforce sequential or state execution.

This lesson contains the following topics:

A. Using Sequential Programming
B. Using State Programming
C. State Machines

Duration: 1 hours 40 min

	Content/Narrative	**Duration**
Slide 1	State the lesson objective. Briefly outline the topics covered in this lesson.	
Slide 2	**A. Using Sequential Programming** **Objective:** Use dataflow to ensure sequential execution of nodes.	
Slide 3	**Why Use Sequential Programming?** • By default, LabVIEW does not force sequential programming. • Highlight that there is no way to enforce execution order of tasks on slides."	
Slide 4	**Flow-Through Parameters** • Use flow-through parameters, such as refnums and error clusters, to control execution order when natural data dependency does not exist. • "Not all LabVIEW nodes have flow-through parameters to ensure data dependency. For example, the One Button Dialog function does not have an error cluster input or output."	

Slide 5

Sequence Structures

- Without flow-through parameters, such as error clusters and refnums, you can use sequence structures to the force execution order.
- “The second frame cannot begin execution until everything in the first frame completes execution.”
- They do not guarantee that the update of terminals will happen before a dialog.

Transition to the next slide: There are appropriate situations for Sequence structures, but in general, Sequence structures are not considered good LabVIEW programming

- They do not take advantage of nodes that do support flow-through parameters.
- They do not guarantee that the update of terminals will happen before a dialog.

Slide 6

Avoid Overuse of Sequence Structures

- Single frame sequence structures are better than multi-sequence frames shown on previous slide.
- In this diagram, the One Button Dialog functions pop-up even if the DAQ Assistant results in an error.

Transition to the next slide: “What can we replace the Sequence structure with?”

Slide 7

Error Case Structures

“Instead, write this VI to enclose the dialog boxes in Case structures, wiring the error cluster to the case selectors.”

Transition to the next slide: “ What if you want a sequence to execute only if a condition is met?”

Slide 8

B. Using State Programming

Objective: The learner will be able to describe the functionality represented by a state transition diagram.

Slide 9

Why Use State Programming

State programming helps you solve the following issues that sequential programming or flow-through parameters do not:

- To change the order of the sequence
- To repeat one item in the sequence more often than other items
- To execute items in the sequence only when certain conditions are met
- To stop the program immediately, rather than waiting until the end of the sequence

Slide 10

State Transition Diagram
A state transition diagram is a type of flowchart that indicates the states of a program and transitions between states

Discuss the definitions of:

- State
- Transition

Slide 11

State Transition Diagram
Highlight the flow chart and how the arrows are the transitions and the states are the circular bubbles on the slides

The solid black circle represents the starting point for the diagram.

The solid black circle with a ring around it represents the end point for the diagram.

Slide 12

C. State Machines

Objective: The learner will be able to recognize and modify the main components of the Simple State Machine project template.

Slide 13

What is a State Machine?

- A state machine implements a state transition diagram or flow chart
- Usually has a start up and shut down state but also contains other states

---

Slide 14

When to Use a State Machine?

- Process tests- Each state represents a segment of the process
- User Interfaces- Different user actions send the user interface into different states.

Slide 15

Multimedia Module: *Building State Machines* 15 min

Location: <Exercises>\LabVIEW Core 1\Multimedia\ Building State Machines

Identify the different components of a state machine and implement transitions between states.

Transition: Now that you have seen the working of a state machine, let's see how you can transition between states depending on user interaction.

Slide 16

Course Project Transition Diagram
Walk students through the five states of the state machine and the transitions between those states.

Slide 17

Exercise 10-1:Weather Station Project 60 min

Goal: Create an application using the Create Project dialog box and the Simple State Machine template.

- Students are given the front panel of a Weather Station application.
- Complete the block diagram by implementing a state machine.
- Use a copy of the Weather Data.ctl and the Temperature Warnings.VI created earlier.
- If they want to use their own versions of the .ctl and .vi files, they can replace the ones in the Weather Station folder.

Slide 18

Discussion: Exercise 10-1: Weather Station Project

How would you change the diagram to add an Initialize and Close state?

1. Add an Initialize item and a Close item to the Weather Station States.ctl typedef.
2. Set the Beginning State to Initialize.
3. Add an Initialize case to the Case structure. Move the File Dialog and Open/Create/Replace function to the Initialize case. Add a transition to Acquisition state.
4. Add a Close case to the Case structure. Move the Close File to the Close case.
5. In the Time Check case, add code to transition to the Close case if the Stop button is pressed.

Slide 19

Event-Based State Machine

- Combines event programming with a State Machine design
- Includes a “Wait on Event” case to processes user-interface events.

Slide 20	**Demo: Simple State Machine Template**	5 min

Location: n/a – Start with a new VI.

1. Launch the **Create Project** dialog box.
2. Show that there are several starting points for applications. You can start new applications from either templates or sample projects.

- Templates are skeleton application where users add code to create their applications.
- Sample projects are complete applications built on top of templates. With sample projects you will likely add, modify, and remove code.
    - o The Single Shot Measurement project is a sample project built on top of the Simple State Machine.

3. Select Simple State Machine and walk through the wizard interface to create a starting application.
4. Show users all the code that is generated.
5. Show users all the help content that is generated.
    - o The blue labels indicate code areas that need modification.
    - o The pale yellow notes are useful for documenting the code.
    - o The project explorer includes extensive documentation on how to modify the template.

Note: The Simple State Machine template uses a Conditional Disable Structure after the Simple Error Handler VI is called.

- o Conditional Disable Structures are beyond the scope of this course. However, you might have students that ask about it.
- o Use this structure when you want to disable specific sections of code on the block diagram based on some user-defined condition.
- o Configure the Conditional Disable symbols for a project from the Project Properties dialog.

6. Enable highlight execution and click on front panel buttons
7. Switch to the block diagram to show how the state transitions occur.

Slide 21 to 25	**Activity 10-1: Lesson Review**	5 min

Refer students to the Participant Guide to answer the questions before showing the slides and discussing the answers with the group

Next Steps…

Objective: Wrap up course and reinforce certification path.

Duration: 5 min

	Content/Narrative	Duration
Slide 1	**Continuing Your LabVIEW Education.**	
Slide 2	**NI LabVIEW Skills Guide**	
Slide 3 and 4	**Continue Your Learning**	
Slide 5	**LabVIEW Certification** Go over the learning path.	
Slide 6	**Thank You!** Ask students to fill out the survey.	