

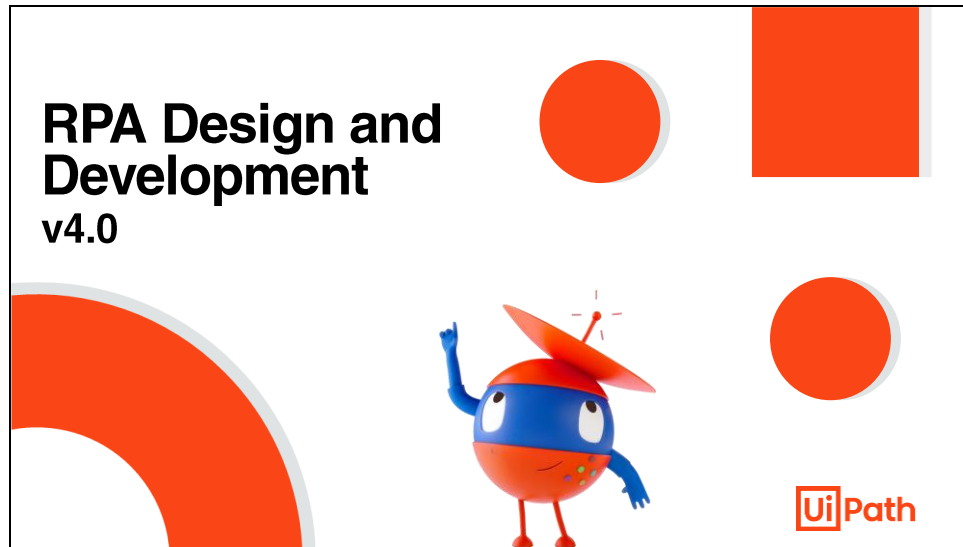
**RPA Design and Development
v4.0**

Student Manual

Lesson 6-Data Manipulation



Slide 1



Welcome to 'RPA Design and Development Course'.




Slide 2



The sixth lesson of this course is Data Manipulation.

Slide 3

Agenda




- 1 Data Manipulation and Its Importance
- 2 String Manipulation
- 3 DataTable Manipulation
- 4 Collection, Its Types and Manipulation

The agenda of this lesson is:


- Data Manipulation and Its Importance
- String Manipulation
- DataTable Manipulation
- Collection, Its Types and Manipulation

Slide 4

Learning Objectives



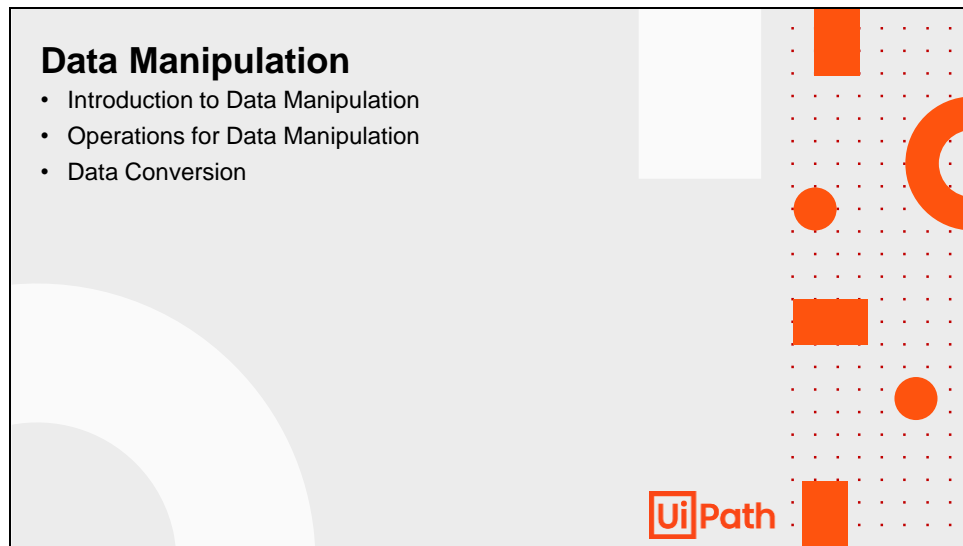
- 1 Describe Data Manipulation and Its Importance
- 2 Explain and Perform String Manipulation
- 3 Explain and Perform DataTable Manipulation
- 4 Explain Collection, Its Types and Manipulation



By the end of this lesson, you will be able to:

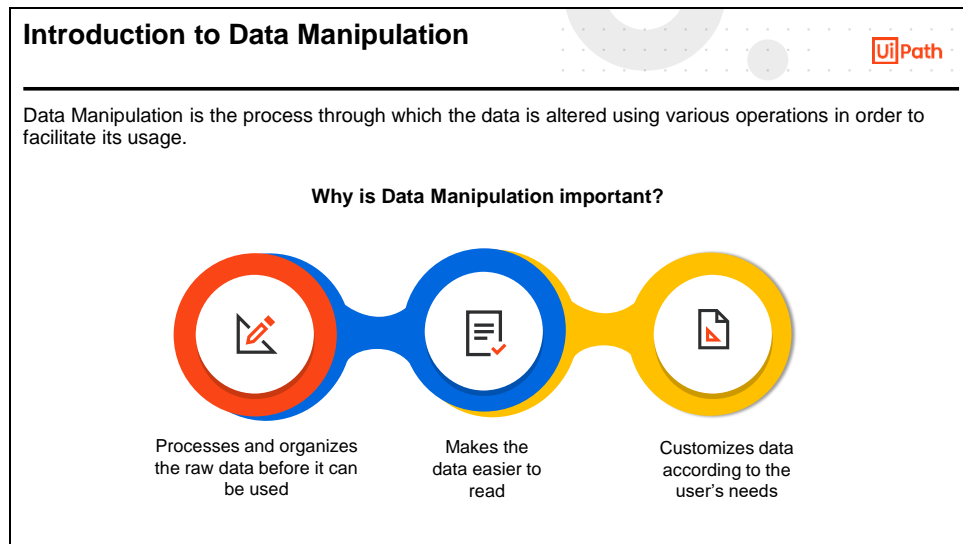
- Describe Data Manipulation and Its Importance
- Explain and Perform String Manipulation
- Explain and Perform DataTable Manipulation
- Explain Collection, Its Types and Manipulation

Slide 5



This section introduces Data Manipulation and Data Conversion.

Slide 6



Data manipulation is the process through which the data is altered using various operations (modifying, structuring, formatting, or sorting). Having raw data is not enough, and it needs to be manipulated as per the user requirement in order to facilitate its usage and increase its management capabilities. Data manipulation allows the user to change data from one form to another by concatenating, converting, extracting, splitting, or segregating to make it ready to be used by RPA in workflows.

Why is Data Manipulation important?

- It makes the data easier to read
- It processes and organizes the raw data before it can be used
- It customizes data according to the user's needs

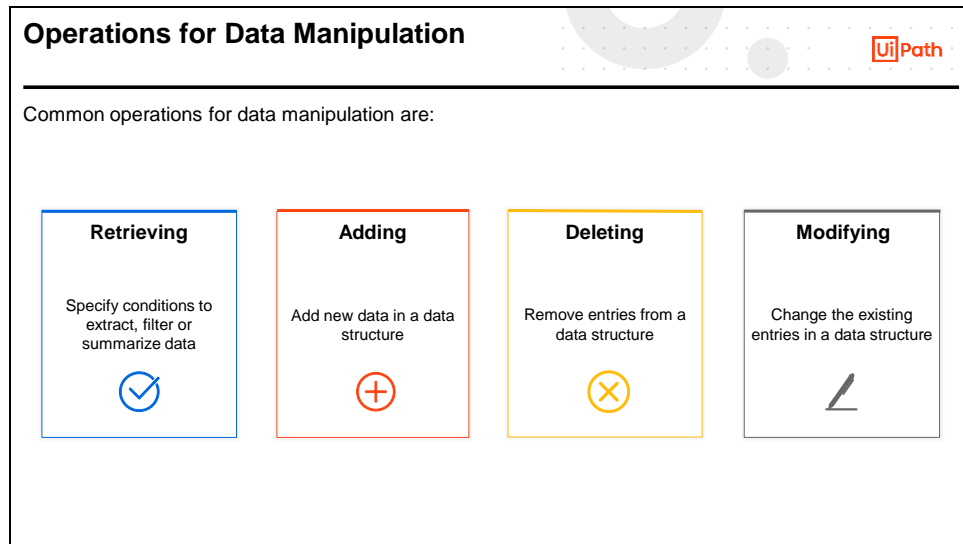
Data of all kinds can be stored in variables, and there are several variable types in Studio that can be classified into three types:

- **Scalar:** Characters, Booleans, or Numbers
- **Collections:** Arrays, Lists, Strings (a collection of characters, and Dictionaries, which are used when extracting data from Orchestrator queues)
- **Tables:** Two-dimensional structures that hold data indexed by rows and columns

Example:

A bulk data (such as a financial database) could be organized in alphabetical order, making individual entries easier to locate. Data manipulation provides to extract only the relevant data and use it in other documents or correlate it with information from other sources.

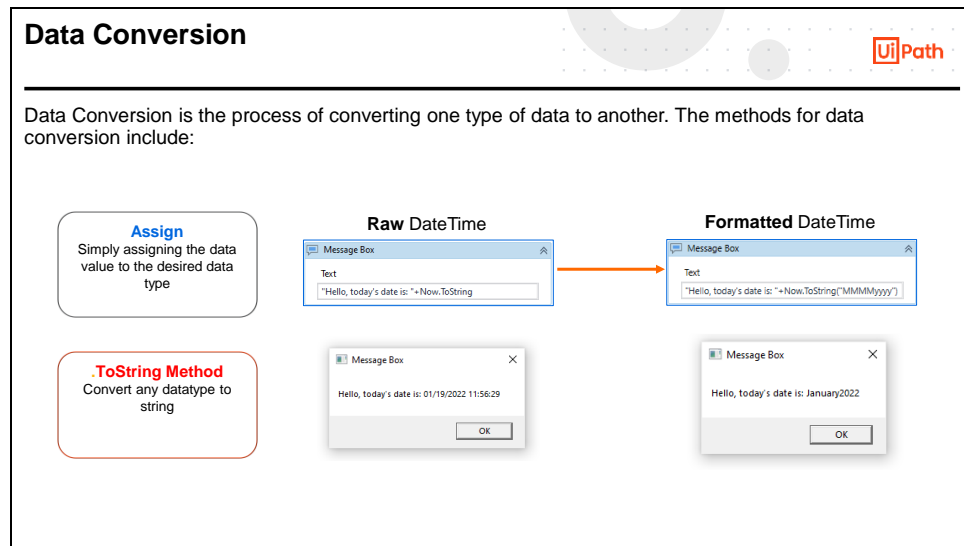
Slide 7



Common operations for data manipulation are:

- Retrieving data: Specify conditions to extract, filter or summarize data
- Adding data: Add new data in a data structure. The user can specify conditions to limit the addition only to filtered entries
- Deleting data: Remove entries from a data structure
- Modifying data: Any type of operation that affects the existing entries in a data structure

Slide 8




Data Conversion is the process of converting one type of data to another.

There are several methods to perform data conversion:

- Assigning a data value to the desired data type. In UiPath, explicit assignment is generally used for data conversion
 - For example: use `integer.Parse("1032")` to convert a string to a number
- Convert any datatype to string using the `.ToString` method
 - Example: The user can convert `DateTime` data type to a string datatype or number to string and vice versa
 - An example of `.ToString` method is shown on the slide

Slide 9

Classroom Exercise



Demonstrate the process to convert an integer to a string using **.ToString** method.

- Ask the user to enter two numbers
- Add both the numbers
- Display the sum along with the text "The output is:" in a message box


Demonstrate the use of **.ToString** method to convert an integer to string. Add two numbers and display the result along with additional text in a message box in string format.

- Insert an **Input Dialog** activity in the designer panel. Enter the text "First Number" in the *Title* and "Enter first number to add" in *Label*. Define a variable called **intFirstNumber** for this activity using the Variables panel. Set Variable Type as **Int32**. Enter this variable in the **Result** property of this activity using its **Properties** panel
- Insert another **Input Dialog** activity below the first **Input Dialog** activity. Enter the text "Second Number" in the *Title* and "Enter second number to add" *Label*. Define a variable called **intSecondNumber** for this activity using the Variables panel. Set Variable Type as **Int32**. Enter this variable in the **Result** property of this activity using its **Properties** panel
- Insert an **Assign** activity below the second **Input Dialog** activity. Press **Ctrl + K** and enter the text **intSum** as variable in the *To* text box, and **intfirstNumber + intsecondNumber** in the adjacent box
- Go to the Variable Panel and set the Variable Type of **intSum** as **Int32**
- Insert the **Message Box** activity below the Assign activity. Enter text "**The output is:** " + **intSum.ToString**
- Run the workflow
 - Enter the first value as **30**, click **OK**.
 - Enter the second value as **40**, click **OK**

- Outcome
 - The result is displayed along with the text
 - Program runs successfully as expected

Slide 10

Practice Exercise



- Build a workflow using the **.ToString** method that converts an integer to a string.
- Ask the user to input their name and age
- Subtract the age of the user with the current year to get the user's year of birth
- Display the name and year of birth in a message box in string format

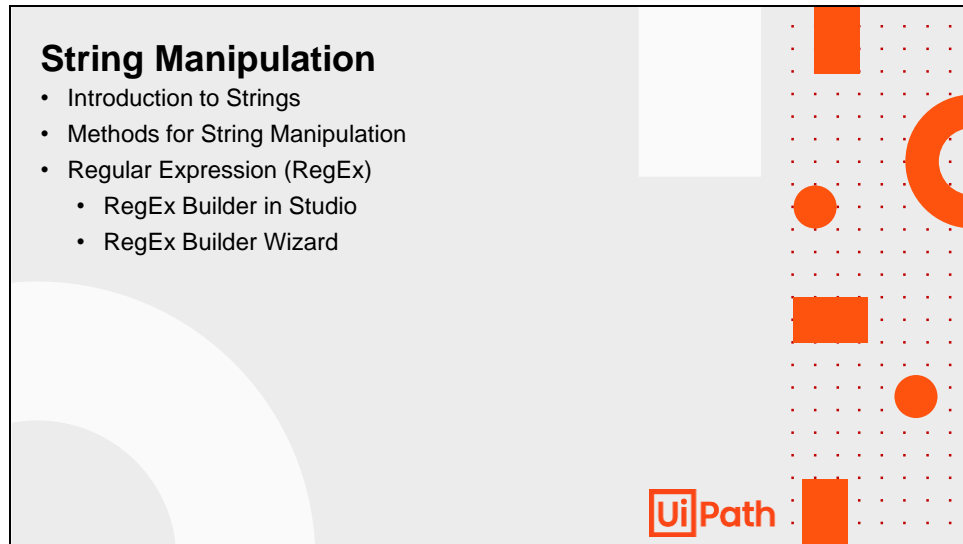
Build a workflow using the **.ToString** method that converts an integer to a string.

- Ask the user to input their name and age
- Subtract the age of the user with the current year to get the user's year of birth
- Display the name and year of birth in a message box in string format

Process Overview

- START
- Use an Input Dialog activity to ask the name and age from the user
- Use an Assign activity and subtract the age of the user from the current year to get the year of birth of the user
- Use a Message Box activity to display: "Hello *User*, you were born in *Year*."
- Replace the *User* and *Year* with name and the year of birth of the user, respectively
- STOP

Slide 11



This section gives an overview of Strings, Its Manipulation and RegEx.

Slide 12

Introduction to Strings

A String is the data type corresponding to text that contains a sequence of text. It is used when a text needs to be captured, processed, displayed or sent between applications.

Name	Variable type	Scope	Default
FirstName	String	Sequence	Enter a VB expression
LastName	String	Sequence	Enter a VB expression
CityName	String	Sequence	Enter a VB expression

Create Variable

A String is the data type corresponding to text that contains any sequence of text. Almost every automation scenario involves the use of strings as they are used when a text needs to be captured, processed, displayed or sent between applications.

Methods for String Manipulation


Some of the operations that can be performed on strings are:

- Concat**
 - Concatenates the string representations of two specified objects
 - Expression: `String.Concat(strVarName1, strVarName2)`
- Contains**
 - Checks whether a specified substring occurs within a string. Returns true or false
 - Expression: `VarName.Contains("text")`
- Format**
 - Converts an entire expression into a string (and inserts them into another text)
 - Expression: `String.Format("{0} is {1}", VarName1, VarName2)`
- IndexOf**
 - Returns the zero-based index of the first occurrence of a character in a string
 - Expression: `VarName1.IndexOf("a")`

Some of the operations that can be performed on strings are:

- **Concat:** Concatenates the string representations of two specified objects
 - Expression: `String.Concat(VarName1, VarName2)`
- **Contains:** Checks whether a specified substring occurs within a string. Returns a boolean value (true or false)
 - Expression: `VarName.Contains("text")`
- **Format:** Converts an entire expression (or value of objects) into a string (and inserts them into another text). Reduces complexity and increases readability
 - Expression: `String.Format("{0} is {1}", VarName1, VarName2)`
- **IndexOf:** Returns the zero-based index of the first occurrence of a character in a string
 - Expression: `VarName1.IndexOf("a")`


Methods for String Manipulation (Contd.)



- Join**
 - Concatenates the elements in a collection and displays them as string
 - Expression: `String.Join("|", CollVarName1)`
- Replace**
 - Replaces all the occurrences of a substring in a string
 - Expression: `VarName.Replace ("original", "replaced")`
- Split**
 - Splits a string into substrings using a given separator
 - Expression: `VarName.Split("|"c)(index)`
- Substring**
 - Extracts a substring from a string using the starting index and the length
 - Expression: `VarName1.Substring(startIndex, length)`

- **Join:** Concatenates the elements in a collection and displays them as a string
 - Expression: `String.Join("|", CollVarName1)`
- **Replace:** Identifies a sequence of characters of the string type in a text and replaces it with a given string
 - Expression: `VarName.Replace ("original", "replaced")`
- **Split:** Splits a string into substrings based on certain criteria set by the user. This criterion could be a space, comma, or full stop
 - Expression: `VarName.Split("|"c)(index)`
- **Substring:** Extracts a substring from a string using the starting index and the length. It is used to isolate or separate a substring from the original string
 - Expression: `VarName1.Substring(startIndex, length)`


Methods for String Manipulation (Contd.)



- ToUpper**
 - Converts lowercase letters to uppercase
 - `strVarName1.ToUpper`
- ToLower**
 - Converts uppercase letters to lowercase
 - `strVarName1.ToLower`
- Parse**
 - Converts a string (text) value to an integer (number) value
 - `Integer.Parse(strVarName1)`
- Trim**
 - Removes blank spaces before and after the string
 - `strVarName1.Trim`

- **ToUpper:** Converts lowercase letters to uppercase
 - Expression: `strVarName1.ToUpper`
- **ToLower:** Converts uppercase letters to lowercase
 - Expression: `strVarName1.ToLower`
- **Parse:** Converts a string (text) value to an integer (number) value
 - Expression: `Integer.Parse(strVarName1)`
- **Trim:** Removes blank spaces before and after the string
 - Expression: `strVarName1.Trim`

Methods for String Manipulation (Contd.)



- Insert**
 - Adds a string to the specified position
 - `strVarName1.Insert(1,strVarName2)`
- Remove**
 - Removes a specified number of characters from a specified position in a string
 - `strVarName1.Remove(1,2)`
- Length**
 - Returns the length of the string
 - `strVarName1.Length`

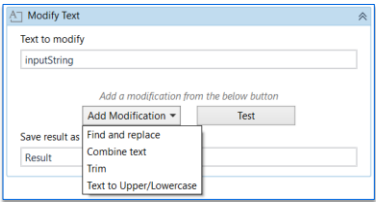
- **Insert:** Adds a string to the specified position
 - Expression: `strVarName1.Insert(1,strVarName2)`
 - First value in () is the position to add
 - * Note that the first character position is 0
 - The second value in () is the string to be added
- **Remove:** Removes a specified number of characters from a specified position in a string
 - Expression: `strVarName1.Remove(1,2)`
 - First value in () is the position where the deletion has to be started
 - * Note that the first character position is 0
 - Second value in (): number of characters to delete
- **Length:** Returns the length of the string
 - Expression: `strVarName1.Length`

Slide 17

Activities for String Manipulation

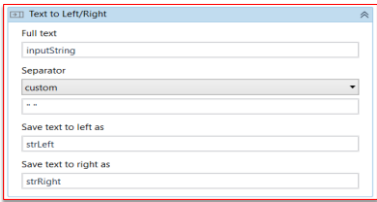
Modify Text

Updates a text value using modifications including find and replace, trim, combine (concatenating) with another text value, and changing to upper/lowercase



Text to Left/Right

Retrieves the text to the left and right of the first occurrence of the indicated subtext



Activities for string manipulations are:

- **Modify Text** - Updates a text value using modifications including find and replace, trim, combining (concatenating) with another text value, and changing to upper/lowercase


To know more, visit: <https://docs.uipath.com/activities/docs/modify-text>

- **Text to Left/Right** - Retrieves the text to the left and right of the first occurrence of the indicated subtext

To know more, visit: <https://docs.uipath.com/activities/docs/text-to-left-right>

Slide 18

Classroom Exercise



Demonstrate the use of **Substring, Concat, Split, Format, and Replace** methods to manipulate strings.


- Use the initial text "I live in Norfolk county in England. It is a great place to visit." for extraction
- Extract the text starting from "Norfolk" using the Substring method
- Display "Thank you for showing interest in touring **extracted text**." in a message box. Replace **extracted text** with the text extracted from the initial text
- Ask for tour budget from the user and display "**budget** is too low to go to **extracted text**." in a message box. Replace **budget** with users' entered value and **extracted text** with the text extracted from the initial text
- Finally, display "But, **budget** is too good to go to Cornwall county in England." in a message box. Replace **budget** with users' entered value. Use Replace method to replace Norfolk with Cornwall county

Demonstrate the use of **Substring, Concat, Split, and Replace** methods to manipulate strings. Demonstrate it by building a workflow that will extract a touring location from a text and display the name along with additional strings in a message box. Also, it will ask the user for the tour budget and suggest an alternative place to visit.

- Insert an **Assign** activity in the designer panel. Press **Ctrl + K** and enter the text **initialText** as a new variable in the *To* text box, and ""I live in Norfolk county in England. It is a great place to visit." in the adjacent box
- Insert another **Assign** activity below the first Assign activity. Press **Ctrl + K** and enter the text **cityName** as a new variable in the *To* text box, and **initialText.Substring(initialText.LastIndexOf("Norfolk"))** in the adjacent box
- Insert a Message Box activity below the second Assign activity. In the text box enter **String.Concat("Thank you for showing interest in touring "+cityName)**
- Insert an **Input Dialog** activity below the Message Box activity. Enter the text "User Budget" in the *Title* and "What is your tour budget?" in double quotes in *Label*. Define a variable called **budget** for this activity using the Variables panel. Set Variable Type as **string**. Enter this variable in the **Result** property of this activity using its **Properties** panel
- Insert an **Assign** activity below the Input Dialog activity. Press **Ctrl + K** and enter the text **message** as a new variable in the *To* text box, and **String.Format("{0} is too low to go to {1}",budget, cityName)** in the adjacent box

- Insert another **Assign** activity below the previous one. Press **Ctrl + K** and enter the text **firstSentence** as a new variable in the *To* text box, and **message.Split("."c).First.ToString** in the adjacent box
- Insert a **Message Box** activity below the **Assign** activity. Enter the variable **firstSentence** in the text box
- Insert another **Message Box** activity below the previous **Message Box** activity. In the text box enter **"But, " + firstSentence.Replace("low","good").Replace("Norfolk","Cornwall")**
- Save and run the workflow
- Outcome
 - County name is extracted and displayed with additional string. Click **OK**
 - It asks for tour budget. Enter **50**. It says budget is too low
 - Click **OK**, now it offers another location to visit. Click **OK**

Practice Exercise



Build a workflow using **Format, Join, IndexOf, Split, and Substring** methods that extracts key information from a text and prints in a different format.

- Use the text "You always wanted to study Automation Training. The materials are available in the following places: UiPath Blog, UiPath Academy." for extraction
- Extract "Automation Training" from the first sentence
- Extract "UiPath Blog" and "UiPath Academy" from the second sentence
- Display "get Automation Training from: UiPath Blog; UiPath Academy" in a message box

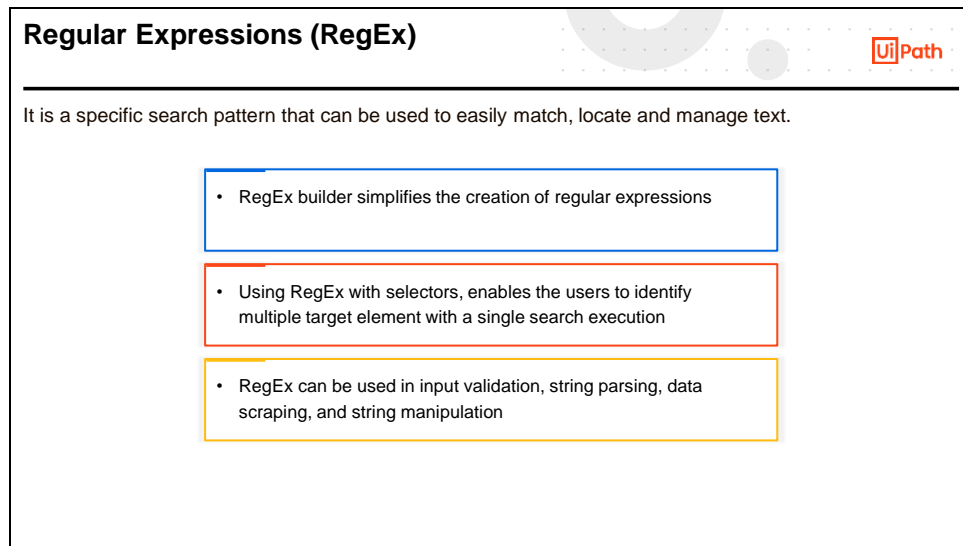
Build a workflow using Format, Join, IndexOf, Split, and Substring methods that extract key information from a text and print in a different format.

- Use the text "You always wanted to study Automation Training. The materials are available in the following places: UiPath Blog, UiPath Academy." for extraction
- Extract "Automation Training" from the first sentence
- Extract "UiPath Blog" and "UiPath Academy" from the second sentence
- Display "get Automation Training from: UiPath Blog; UiPath Academy" in a message box

Process Overview

- START
- Use an Assign activity for the initial value of the message string variable: "You always wanted to study Automation Training. The materials are available in the following places: UiPath Blog, UiPath Academy."
- Create a new String variable **study** and use a succession of String methods to assign the course from the query: **message.Split("."c).First.ToString.Substring(message.LastIndexOf("study"))**
 - **Split("."c).First.ToString** extracts the first sentence of the String and converts it to a String
 - **Substring(message.LastIndexOf("study"))** extracts the Substring starting from "get"

- Create a new List variable **places** and use a succession of String methods to assign the places from the query:
message.Split("."c)(1).ToString.Split(":"c).Last.ToString.Split(", "c).ToList
 - **message.Split("."c)(1).ToString** extracts the second sentence of the String and converts it to a String
 - **Split(":"c).Last.ToString** splits the remaining string and keeps only the last part of it
 - **Split(", "c).ToList** takes each string separated by comma and adds it as an element in the List variable
- Use a Message Box activity to display output using this expression:
String.Format("{0} from: {1}", study ,String.Join(";", places))
 - **String.Join** is used to extract each element in the “places” List variable and display them
- STOP



Regular Expressions (Regex)

It is a specific search pattern that can be used to easily match, locate and manage text.

- Regex builder simplifies the creation of regular expressions
- Using Regex with selectors, enables the users to identify multiple target element with a single search execution
- Regex can be used in input validation, string parsing, data scraping, and string manipulation

Regular Expression (Regex) is a specific search pattern that can be used to match, locate and manage text easily.

Studio contains a Regex builder that simplifies the creation of regular expressions. Using the Regex search capabilities in selectors enables the users to identify multiple target elements with a single search execution. Without Regex, multiple selectors would have been built to identify each target element.

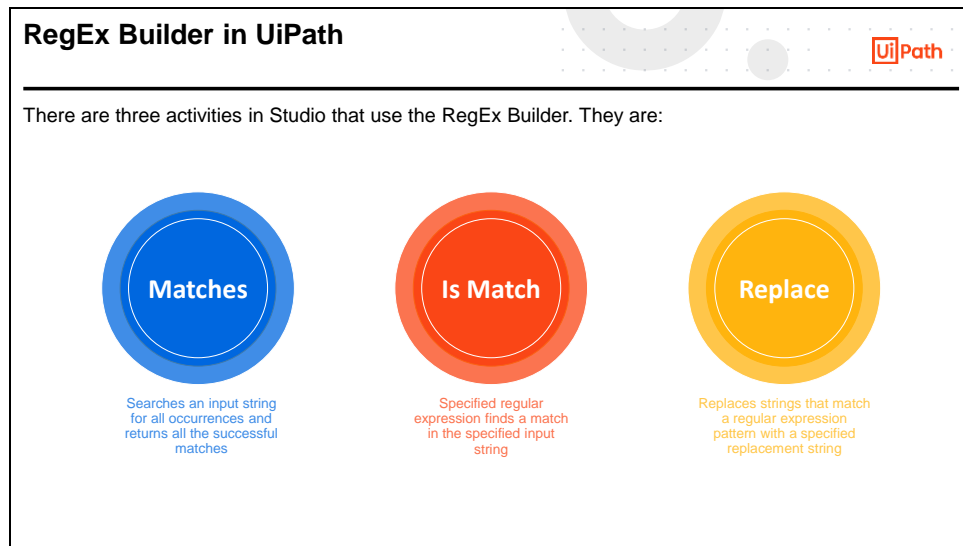
Regex can be used in:

- Input validation
- String parsing
- Data scraping
- String manipulation

Example:

Regex can be used for retrieving pieces of text that follow a certain pattern, such as extracting phone numbers that start with a certain digit.

To know more, visit: <https://docs.uipath.com/studio/v2021.10/docs/regex-search>



There are three activities in Studio that use the RegEx Builder. They are:

- **Matches:** Searches an input string for all occurrences and returns all the successful matches for the given expression
 - Can be used to retrieve all the entries and use them further

To know more, visit: <https://docs.uipath.com/activities/docs/matches>

- **Is Match:** Indicates whether the specified regular expression finds a match in the specified input string and returns only a True/False value
 - Can be used as a Condition for another activity

To know more, visit: <https://docs.uipath.com/activities/docs/is-match>

- **Replace:** Replaces strings that match a regular expression pattern with a specified replacement string
 - Can be used for data quality purposes

To know more, visit: <https://docs.uipath.com/activities/docs/replace>

Slide 22

RegEx Builder Wizard

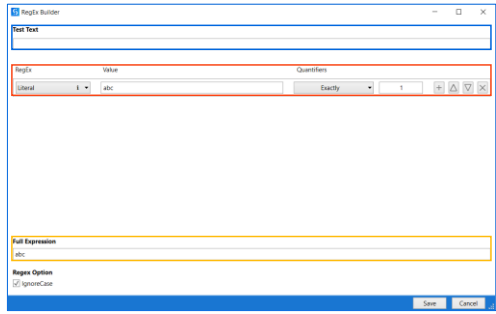
Eases the process of building and testing Regular Expression search criteria. It can be opened from the body of any of the three activities (Matches, Is Match, and Replace).

Test Text
A text editor where the user can test the chosen search criteria against the text on which RegEx is applied

Type, Value, and Quantifiers

- **Type:** Allows searching for a given text or one expression from many.
- **Value:** Contains exactly the text that needs to be retrieved.
- **Quantifiers:** A dropdown list that enables the user to select the type of results that should be displayed

Full Expression
Displays the current RegEx expression in its raw form



The RegEx Builder wizard is created to ease the process of building and testing Regular Expression search criteria. The RegEx Wizard can be opened from the body of any of the three activities (Matches, Is Match, and Replace).

It contains a text editor called **Test Text**. Here, the user can test the chosen search criteria against the text on which RegEx is applied. Below the text editor, the user can choose the Type, Value, and Quantifiers of the RegEx expressions. Doing so highlights the findings in the Test Text editor.

It simplifies building regular expressions by allowing the configuration of Type, Value and Quantifiers. Once a condition is finalized, another one can be added. If there is more than one condition, the order in which they are applied can be configured as well.

- **Type:** Allows searching for a given text or one expression from many. It can be configured to search only at the beginning or at the end, and it offers also pre-built expressions for Emails, URLs, US dates or US phone numbers, which are very useful in the case of standardized data

Some examples are:

- Literal
- Digit
- One of
- Not one of


- Anything
- Any word character
- Whitespace
- Starts with
- **Value:** Contains exactly the text that needs to be retrieved
- **Quantifiers:** A dropdown list that enables the user to select the type of results that should be displayed
 - Exactly: To select an exact amount of consecutive occurrences the user wants to find.
 - For example, if the text being searched for is hello and the Quantifiers is set to Exactly 2, the wizard finds any occurrences of hellohello in the Test Text box
 - Any (0 or more): Highlights any number of consecutive found matches, starting from 0
 - At least one (1 or more): Highlights any number of consecutive found matches, starting from 1
 - Zero or one: Only highlights a single consecutive occurrence of the term
 - Between X and Y times: Highlights the amount of consecutive occurrences selected
 - For example, searching for hello and selecting Between 2 and 3 times only highlights hellohello and hellohellohello

The **Full Expression** text box at the bottom of the wizard displays the current RegEx expression in its raw form.

To know more, visit: <https://docs.uipath.com/activities/docs/regex-builder-wizard>

Slide 23

Classroom Exercise



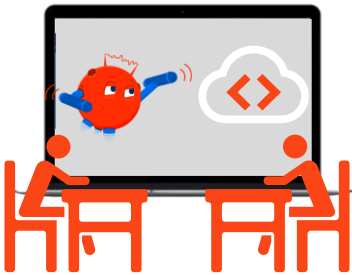
Demonstrate the use of **Regex Builder** in order to extract email IDs from a text.

- Use the text "My name is Joe. My email is joe@mail.com, and my father's email is jack@mail.com, and my uncle's email is jay@mail.com" for extraction
- Extract email ids from the text using Regex Builder wizard
- Store the email ids in an MS Word file

Demonstrate the use of **Regex Builder** in order to extract email IDs from a text. Demonstrate it by building a workflow that can extract email ids from a text, and display in the Output panel.

- Insert an **Assign** activity in the designer panel. Press **Ctrl + K** and enter the text **message** as a variable in the *To* text box, and in the adjacent box enter "My name is Joe. My email is joe@mail.com, and my father's email is jack@mail.com, and my uncle's email is jay@mail.com."
- Insert a **Matches** activity below the Assign activity. Click on the **Configure Regular Expression** button within the Matches activity. In the **RegEx Builder** wizard, go to the **RegEx** column and select **Email** from the drop down. In the **Quantifiers** column, select **Any** from the drop down
- Go to the Properties panel of the Matches activity. In the **Input** property, enter the existing variable **message**. In the **Result** property, press **Ctrl + K** on your keyboard, and enter a new variable **emailIDs** after **Set Var**:
- Insert a **For Each** activity after the Matches activity. Leave the text **item** in the first box as it is, and in the *Vb Expression* text box, enter the variable **emailIDs**
- In the Body section of the For Each activity, enter a **Write Line** activity, and enter **item.ToString** in the text area
- Save and run the workflow
- Go to the Output panel
- Outcome
 - All email ids from the text are extracted and listed here

Practice Exercise



- Build a workflow using **Split** and **Contains** methods that extract sentences containing "RPA" from a paragraph.
- Store a paragraph in a string variable using an Assign activity
- Store all sentences from the text in an array using a Split method
- Loop through each sentence and identify sentences containing "RPA" using the Contains method
- Store all identified sentences in an MS Word file

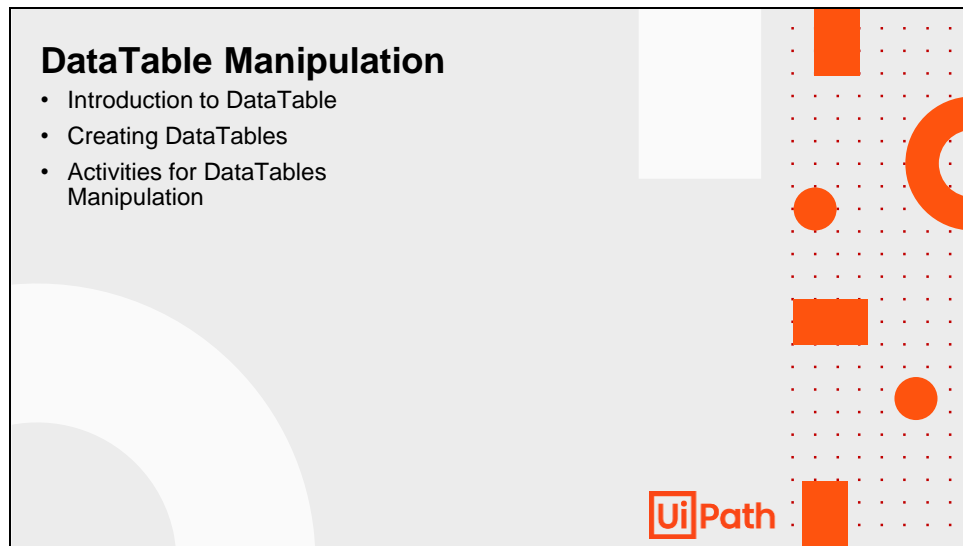
Build a workflow using **Split** and **Contains** methods that extract sentences containing "RPA" from a paragraph.

- Store a paragraph in a string variable using an Assign activity
- Store all sentences from the text in an array using a Split method
- Loop through each sentence and identify sentences containing "RPA" using the Contains method
- Store all identified sentences in an MS Word file

Process Overview

- START
- Use an Assign activity to store a paragraph in a string variable called **newText**
- Use **newText.Split(".",c)** and store all sentences in an array called **newSentence**
- Use a For Each activity and iterate through each item in the array **newSentence**
- Use an If activity within the For Each activity to identify sentence that contains the string "UiPath in it. Use **item.Contains("RPA")** as condition
- Use a Type Into activity to store result in an MS Word file
- STOP

Slide 25

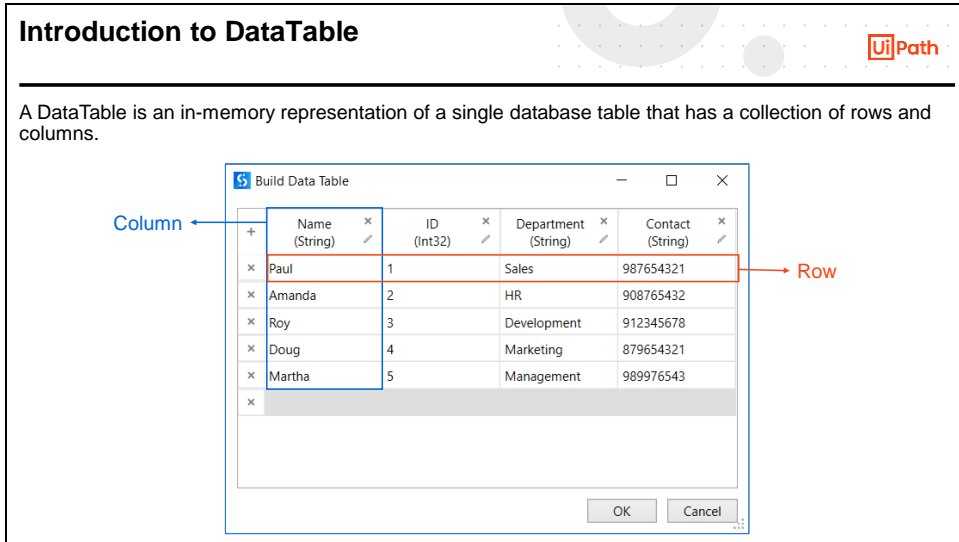


This section gives an overview of DataTable and Its Manipulation.

Slide 26

Introduction to DataTable

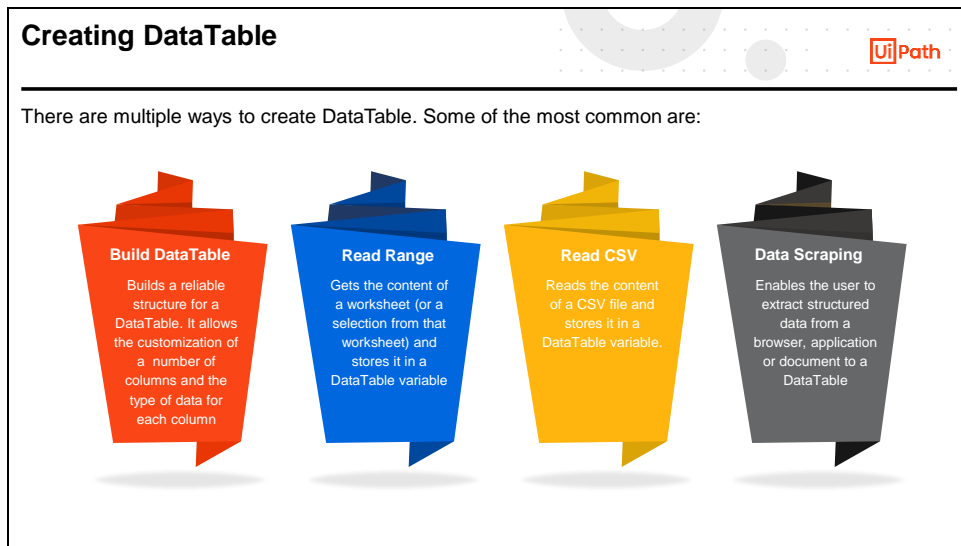
A DataTable is an in-memory representation of a single database table that has a collection of rows and columns.



	Name (String)	ID (Int32)	Department (String)	Contact (String)
×	Paul	1	Sales	987654321
×	Amanda	2	HR	908765432
×	Roy	3	Development	912345678
×	Doug	4	Marketing	879654321
×	Martha	5	Management	989976543
×				

A DataTable is an in-memory representation of a single database table that has a collection of rows and columns. DataTable stores data as a simple spreadsheet with rows and columns, so that each piece of data can be identified based on its unique column and row coordinates.


In DataTables, columns are identified through capital letters and rows through numbers.



There are multiple ways to create DataTables. Some of the most common are:

- **Build DataTable:** Builds a reliable structure for a DataTable using a dedicated window. This activity allows the customization of a number of columns and the type of data for each column. The user can configure each column with specific options like allow null values, unique values, auto-increment (for numbers), default value, and length (for strings)
- **Read Range:** Reads the data from an existing file. It gets the content of a worksheet (or a selection from that worksheet) and stores it in a DataTable variable, which can be created from the Properties panel using Ctrl + K
- **Read CSV:** Reads the content of a CSV (comma-separated values) file and stores it in a DataTable variable. It is similar to Read Range, with the difference that it works for CSV files as input
- **Data Scraping:** Enables the user to extract structured data from a browser, application, or document to a DataTable

Classroom Exercise




Demonstrate the use of **Build Data Table** and **Output Data Table** activities to store phone numbers in a data table and print it in the output panel in string format.

- Create a data table and store five phone numbers in it
- Print the phone numbers in string format in the Output panel

Demonstrate the use of Build Data Table and Output Data Table activity to print a data table in string format. Demonstrate it by building a workflow that stores five phone numbers in a data table and prints it in the output panel in string format.

- Insert a **Build Data Table** activity in the designer panel. Click the **Data Table** button within the activity. In the pop-up table, rename the column header of the first column from “Column1” to “Phone Numbers”. Change its **Data Type** to **Int32**. Enter five phone numbers of your choice. Now, delete the second empty column and click **OK**
- Go to the **Properties** panel of the **Build Data Table** activity. In the Output property, press **Ctrl + K** and enter a data table variable **dt_PhoneNumbers**
- Insert **Output Data Table** activity below the **Build Data Table** activity. Go to its Properties panel, and in the **Input** property enter the variable **dt_PhoneNumbers**. In the Output property, press **Ctrl + K** and enter a new string variable called **stringPhoneNumbers**
- Insert a **Write Line** activity below the **Output Data Table** activity. Enter **stringPhoneNumbers** in the text area
- Save and run the workflow
- Open the **Output** panel
- Outcome
 - All the phone numbers are listed here as strings

Activities for DataTable Manipulation



Some of the manipulations that can be done on DataTable are:

Add Data Column

- Adds a column to an existing DataTable variable
- The input data can be of DataColumn type, or the column can be added empty by specifying the data type and configuring the options

Add Data Row

- Adds a new row to an existing DataTable variable
- The input data can be of DataRow type or can be entered as an Array Row by matching each object with the data type of each column

Merge Data Table

- Appends a specified DataTable to the current DataTable
- The operation is simpler than the Join Data Type activity, as it has 4 predefined actions to perform over the missing schema

Some of the manipulations that can be done on DataTable are:

- **Add Data Column:** Adds a column to an existing DataTable variable. The input data can be of DataColumn type, or the column can be added empty by specifying the data type and configuring the options (allowing null values, requesting unique values, auto-incrementing, default value, and maximum length)
- **Add Data Row:** Adds a new row to an existing DataTable variable. The input data can be of DataRow type or can be entered as an Array Row by matching each object with the data type of each column
- **Merge Data Table:** It is used to merge a specified DataTable with the current DataTable. The operation is simpler than the Join Data Type activity, as it provides four types of actions to take when merging – adding, ignoring, returning an error, and manual input (Addwithkey option)

Activities for DataTable Manipulation (Contd.)

Some of the manipulations that can be done on DataTable are:

Lookup Data Table

- Allows searching for a provided value in a specified DataTable
- It returns the RowIndex at which it was found or can be configured to return the value from a cell with given coordinates

Filter Data Table

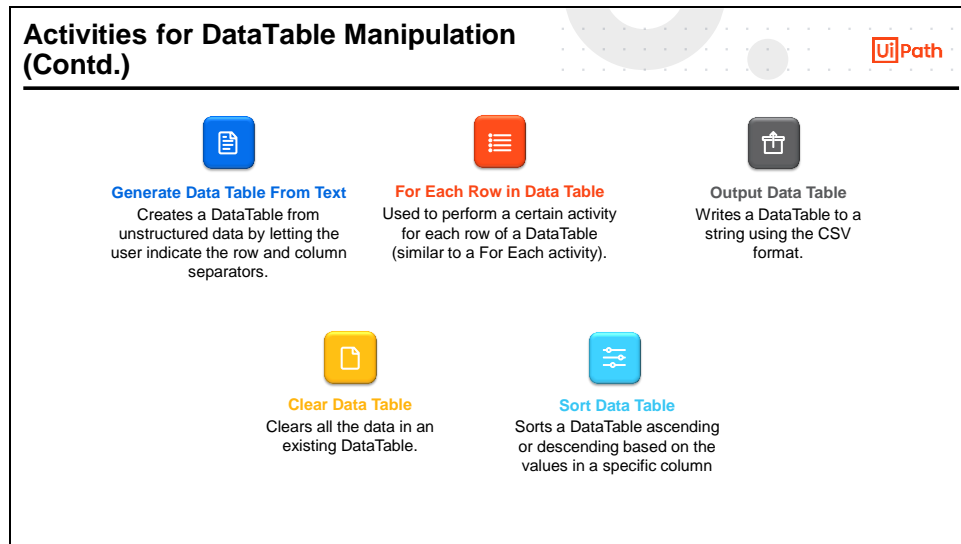
- Allows filtering a DataTable through a Filter Wizard, using various conditions
- It can be configured to create a new DataTable for the output of the activity or to keep the existing one and filter out entries

Join Data Tables

- Combines rows from two tables by using values common to each other
- It is one of the most useful activities in business scenarios, where working with more than one Data Table is very common

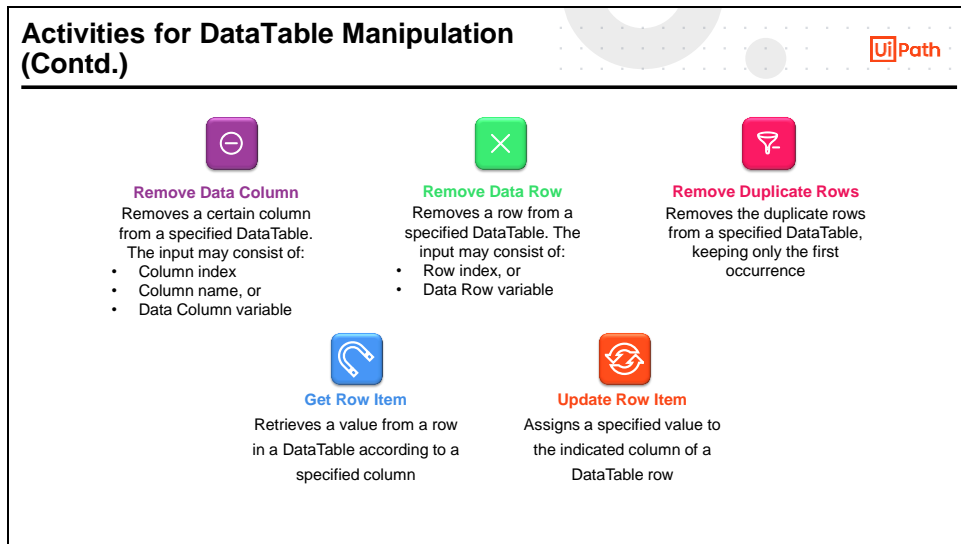
Some of the manipulations that can be done on DataTable are:

- **Lookup Data Table:** It allows searching for a provided value in a specified DataTable and returns the RowIndex at which it was found or can be configured to return the value from a cell with given coordinates (RowIndex and Target Column)
- **Filter Data Table:** It allows filtering a DataTable through a Filter Wizard, using various conditions. This activity can be configured to create a new DataTable for the output of the activity or to keep the existing one and filter out (delete) the entries that do not match the filtering conditions
- **Join Data Tables:** It combines rows from two tables by using values common to each other. It is one of the most useful activities in business scenarios, where working with more than one Data Table is very common



- **Generate Data Table From Text:** Creates a DataTable variable from unstructured data by letting the user indicate the row and column separators. This option is extremely useful when data is captured from scanned documents or web scraping. In order to be able to process the data, the creation of a DataTable is the first step. The activity allows the definition of separators (any string character) but also allows the identification of elements using their position on the screen (useful when the data source is OCR)
- **For Each Row in Data Table:** It is similar to the For Each loop as it iterates through all the rows in a DataTable and performs the same action. This can be very useful for manipulations that have to be done individually for each row and cannot be grouped at the column level (for example, verification of the relation between two values stored in two columns)
- **Output Data Table:** It is used to write a specified DataTable into a text (string variable) in a CSV format. This can serve as an intermediate step in a process or as a final step when, after several manipulations, a DataTable contains only several values
- **Clear Data Table:** It clears all the data entries in a DataTable. It can be very useful with DataTables that are used as intermediary tables for data moved from one table to another
- **Sort Data Table:** It is used to sort a DataTable using a specified column as the criterion. Besides indicating which column to use in this respect, the user can choose whether the sorting will be done ascending or descending

To know more, visit: <https://docs.uipath.com/activities/docs/programming-system>




- **Remove Data Column:** It is used to remove a DataColumn from a specified DataTable. The Columns can be identified by their name and index number
- **Remove Data Row:** It is used to remove a DataRow from a specified DataTable. The Rows can be identified by their index number or by inputting the DataRow as an object
- **Remove Duplicate Rows:** It is used to remove the duplicate rows while keeping only the first occurrence
- **Get Row Item:** It is used to retrieve a value from a row in a DataTable according to a specified column
- **Update Row Item:** It is used to assign a specified value to the indicated column of a DataTable row

To know more, visit: <https://docs.uipath.com/activities/docs/programming-system>

Slide 33

Classroom Exercise



Demonstrate the use of the **Sort Data Table** activity to sort a table.


- Create a data table to store names of five students and their ages
- Sort the data table based on students' names in alphabetical order from A to Z
- Print students' name and their age in the Output panel in a string format

Demonstrate the use of the **Sort Data Table** activity to sort a table. Demonstrate it by building a workflow that creates a data table containing students' name and their age. It then sorts the table in alphabetical order based on students' names from A to Z, and then prints students' name and their ages in the Output panel in a string format.

- Insert a **Build Data Table** activity in the designer panel. Click on the **DataTable** button within the Build Data Table activity. Insert five student names in the order **Jack, Augustine, Brandon, Noice, and Drake** in the first column. In the second column enter students' age as **24, 28, 21, 22, and 20**. Click **OK** to save the table
- Go to the **Properties** panel of the **Build Data Table** activity. In the Output property, press **Ctrl + K** on your keyboard and enter a new variable called **dt_Students**
- Insert **Sort Data Table** activity below the Build Data Table activity. Go to its **Properties** panel, and enter the data table name **students** in the **Input** property. In the **Output** property, press **Ctrl + K** on your keyboard, and enter a new variable called **dt_SortedTable**. In the **Index** property, enter value **0**. In the **Order** property, choose **Ascending**
- Now, insert an **Output Data Table** activity below the Sort Data Table activity. Go to its **Properties** panel, and enter **dt_SortedTable** in the Input property. In the **Output** property, press **Ctrl + K** on your keyboard, and enter a new variable called **SortedByName**
- Insert a **Write Line** activity below the Output Data Table activity. Enter the variable **SortedByName**
- Save and run the workflow


- Go to the Output panel
- Outcome
 - All the names of the students are listed in ascending order along with their ages

Practice Exercise



Build a workflow using **DataTable** activities to join two library databases using matching student ID and display output in a message box.

- Create a data table variable and populate it with student ID and the name of students
- Create another data table variable, and populate it with student ID and book names
- Join both the data tables based on matching student ID
- Remove student ID column and sort the final data table as per student names in alphabetical order from A to Z
- Display the final data table containing student and book names in a message box as a string



Build a workflow using **DataTable** activities to join two library databases using matching student ID and display output in a message box.

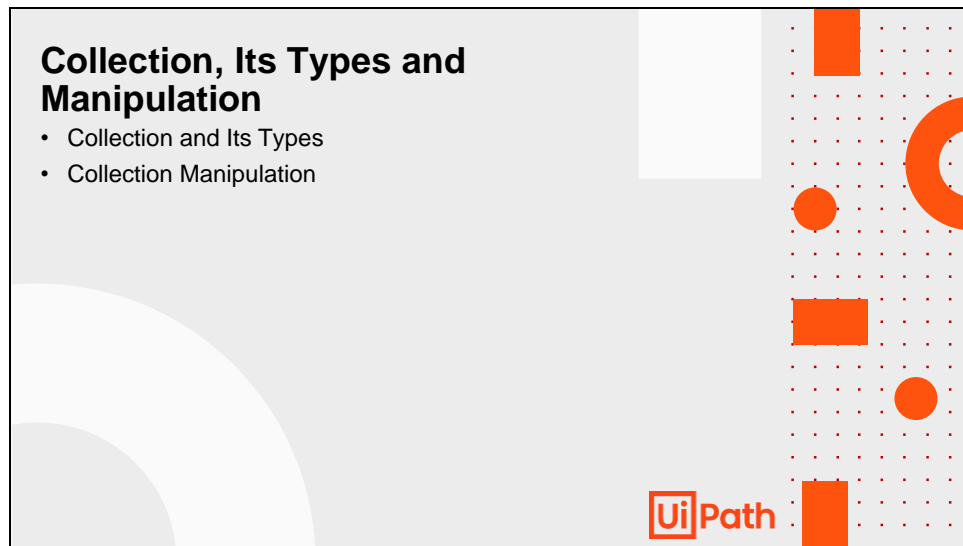
- Create a data table variable and populate it with student ID and the name of students
- Create another data table variable, and populate it with student ID and book names
- Join both the data tables based on matching student ID
- Remove student ID column and sort the final data table as per student names in alphabetical order from A to Z
- Display the final data table containing student and book names in a message box as a string

Process Overview

- START
- Use two Build Data Table activities to create two tables. Store them in two DataTable variables called **dt_users** and **dt_overdueBooks**
 - dt_users** variable contain ID of students and name of user as string
 - dt_overdueBooks** variable contain ID of students and name of books as string
- Use a Join Data Table activity. Choose the Inner type for the Join activity. Write the two column names to be used as Join criterion and create a new data table variable to store the output called **dt_borrowedBooks**
- Use a Remove Data Column activity to delete duplicate column – student ID – by specifying its index

- Use a Sort Data Table activity to sort the data table based on the name of students in alphabetical order from A to Z
- Use an Output Data Table activity to print the content of the data table to a String variable
- Use the Message Box activity to display the output
- STOP


Slide 35




This section gives an overview of Collection, Its Types and Manipulation.

Collection and Its Types


A collection is used to represent a set of similar data type items in a single unit which is used for grouping and managing the objects. Different types of collections are:



Arrays



Lists



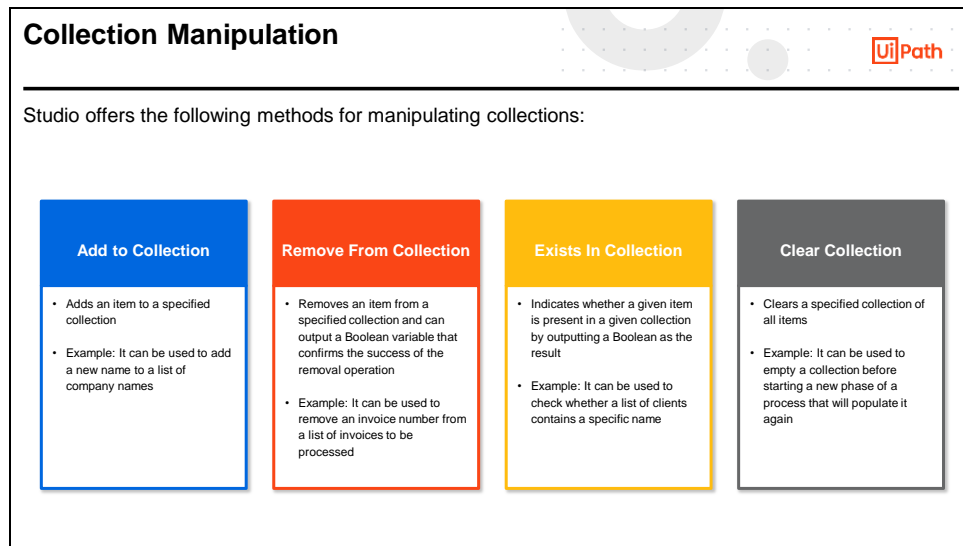
Dictionaries

A collection is used to represent a set of similar data type items in a single unit which is used for grouping and managing the objects. It is useful in cases when the user wants to extract the data in the Excel spreadsheet and orchestrator queues.

The different types of collections are:

- Arrays
- Lists
- Dictionaries

Arrays have been discussed previously, and lists and dictionaries are discussed in the coming slides.



Collection Manipulation

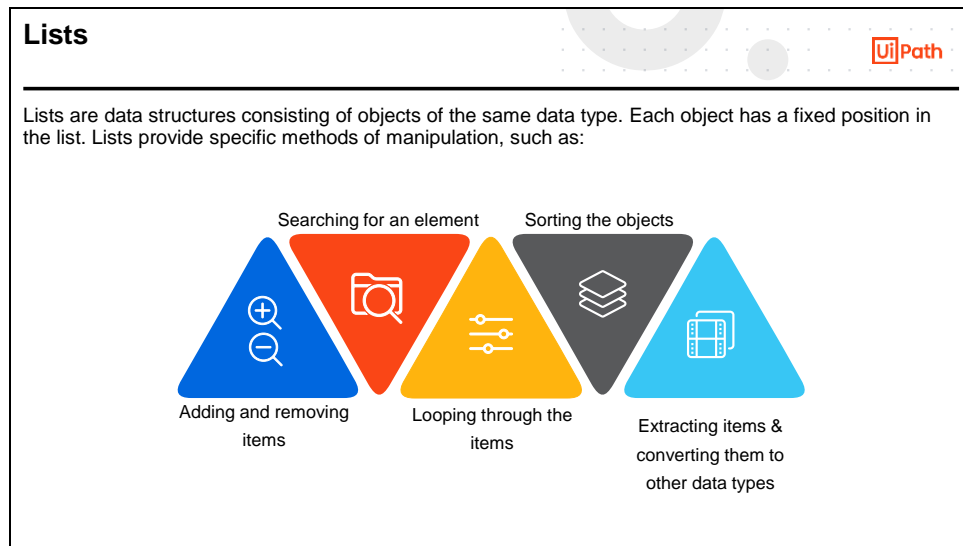
Studio offers the following methods for manipulating collections:

Add to Collection	Remove From Collection	Exists In Collection	Clear Collection
<ul style="list-style-type: none">Adds an item to a specified collectionExample: It can be used to add a new name to a list of company names	<ul style="list-style-type: none">Removes an item from a specified collection and can output a Boolean variable that confirms the success of the removal operationExample: It can be used to remove an invoice number from a list of invoices to be processed	<ul style="list-style-type: none">Indicates whether a given item is present in a given collection by outputting a Boolean as the resultExample: It can be used to check whether a list of clients contains a specific name	<ul style="list-style-type: none">Clears a specified collection of all itemsExample: It can be used to empty a collection before starting a new phase of a process that will populate it again

Studio offers the following methods for manipulating collections:

- **Add To Collection:** Adds an item to a specified collection. It is equivalent to List.Add()
 - Example: It can be used to add a new name to a list of company names
- **Remove From Collection:** Removes an item from a specified collection and can output a Boolean variable that confirms the success of the removal operation
 - Example: it can be used to remove an invoice number from a list of invoices to be processed
- **Exists In Collection:** Indicates whether a given item is present in a given collection by outputting a boolean as the result
 - Example: it can be used to check whether a list of clients contains a specific name
- **Clear Collection:** Clears a specified collection of all items
 - Example: it can be used to empty a collection before starting a new phase of a process that will populate it again

Slide 38



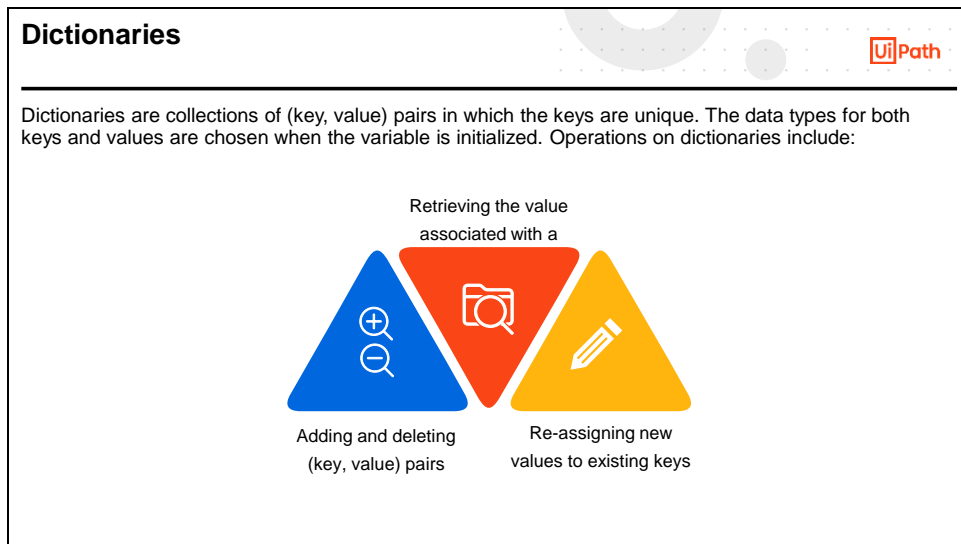
Lists are data structures consisting of objects of the same data type (for example, string or integer). Each object has a fixed position in the list; thus, it can be accessed by index. While arrays are fixed-size structures for storing multiple objects, lists allow the user to add, insert and remove items. Lists are one of the simplest and most used collection data types.

Lists can store large numbers of elements - names, numbers, time coordinates, etc.

Lists provide specific methods of manipulation, such as:

- Adding and removing items
- Searching for an element
- Looping through the items (and performing certain actions on each)
- Sorting the objects
- Extracting items and converting them to other data types

Slide 39



A dictionary is a collection of words and their meanings/definitions. Dictionaries consist of (key, value) pairs in which the keys are unique. Here, the key is equivalent to words, and value is equivalent to meanings/definitions.

Example: Address Book in mobile phone, where each name has corresponding data (phone number(s) & email).

The data types for both keys and values have to be chosen when the variable is initialized. Data types in dictionaries can be any of the supported variables.

The operations that are most often associated with dictionaries are:

- Adding and deleting (key, value) pairs
- Retrieving the value associated with a key
- Re-assigning new values to existing keys

Slide 40



Dictionaries in Studio can be used in the following ways:

- Initializing a dictionary
- Adding an item to an existing dictionary
- Removing an item from a dictionary
- Retrieving dictionary items

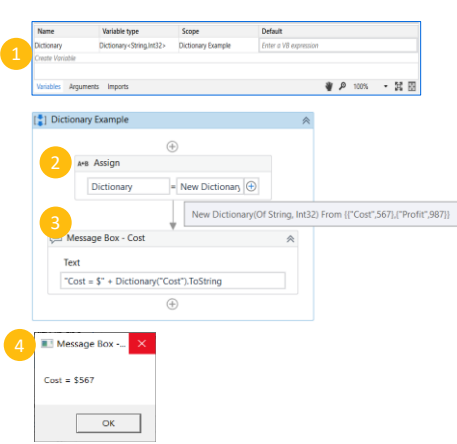
The methods that can be adopted to do all the above are discussed in the subsequent slides.

Slide 41

Methods for Working with Dictionaries

Initializing

1. Create a variable of type Dictionary in the Variables panel
2. Drag and drop an Assign activity in the Designer panel
 - Enter Dictionary in the *To* textbox.
 - Initialize the dictionary variable in the *Value* textbox using **new Dictionary (Of String, Int32) From {{key, value}, {key, value}}**, if the key is String and the value is Integer
3. Insert a Message Box activity to display the added key and value pair
4. Execute the workflow. The Value of the first Key displays in the message box



The screenshot illustrates the steps in the UiPath IDE:

- Step 1:** The Variables panel shows a variable named 'Dictionary' of type 'Dictionary<String, Int32>' with scope 'Dictionary Example' and default value 'Enter a VB expression'.
- Step 2:** An 'Assign' activity is added to the workflow. The 'To' field is set to 'Dictionary' and the 'Value' field contains the expression: `New Dictionary(Of String, Int32) From [{"Cost":567}, {"Profit":987}]`.
- Step 3:** A 'Message Box' activity is added below the Assign activity. Its 'Text' field is set to `"Cost = $" + Dictionary("Cost").ToString`.
- Step 4:** The final output shows a message box titled 'Message Box - Cost' with the text 'Cost = \$567'.

Initializing:

- Create a variable of type Dictionary in the Variables panel
- Drag and drop an Assign activity in the Designer panel
 - Enter Dictionary in the *To* textbox
 - Initialize the dictionary variable in the *Value* textbox using **new Dictionary (Of String, Int32) From {{key, value}, {key, value}}**, if the key is String and the value is Integer. The key-value pair can be string, int or object datatypes
- Insert a Message Box activity to display the added key and value pair
- Execute the workflow. The Value of the first Key displays in the message box

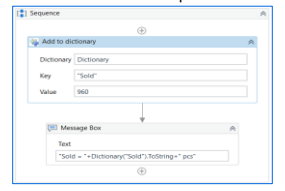
In the example on the slide, the key is 'Cost', and the value is '984252'. When the workflow is executed, it displays the Cost '984252' in the message box.

Slide 42

Methods for Working with Dictionaries (Contd.)

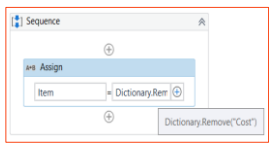
Adding

- Install the package Microsoft.Activities.Extension.
- Drag and drop the Add to Dictionary activity
- Enter the dictionary variable, key, and value in their respective fields



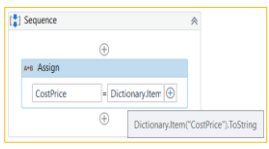
Removing

- VarName.Remove(Key) – removes an item from the Dictionary. It can be used in an 'Assign' activity



Retrieving

- VarName.Item(Key) – returns the Dictionary item by its key



Adding:

- Install the package Microsoft.Activities.Extension
- Drag and drop the Add to Dictionary activity
- Enter the dictionary variable, key, and value in their respective fields

Removing:


- VarName.Remove(Key): Removes an item from the dictionary. It can be used in an 'Assign' activity

Retrieving:

- VarName.Item(Key): Returns the dictionary item by its key
- VarName.Count: Returns an Int32 value of the number of Dictionary items
- VarName.ContainsKey(Key): Checks if the item with the given key exists in the Dictionary and returns a Boolean result
- VarName.TryGetValue(Key, Value): Checks if an item with a given key exists in the Dictionary and returns a Boolean result and the value if found

Slide 43

Classroom Exercise



Demonstrate the steps to sort a list in reverse order and print the first five items from the list in a message box.

- Create a list of ten names of people
- Sort the names in reverse alphabetical order from Z to A.
- Extract the first five names from the list
- Display the extracted names in a message box


Demonstrate the steps to sort a list in reverse order and print the first five items from the list in a message box. Demonstrate it by building a workflow that uses a list of student names in random alphabetical order. It sorts and reverses the list to display the first five student names in a message box.


- Drag and drop a **Sequence** activity in the designer panel. In the **Variables Panel**, create a list variable called **studentList**. Under the **Variable Type** column of **studentList**, choose **Browse for Types** from the drop down list. In the popup window, search **List** and select **List<T>** under **System.Collections.Generic**. Now, click the drop down menu that appears at the top of the popup window, and select **String**. Click **OK** at the bottom of the window to confirm your selection
- Under the **Default** column of **studentList**, initialize the variable by entering the expression **new List(of String) from {"Jack", "Anna", "Ben", "Zed", "Tom", "Harry", "Dick", "Sen", "Eggy", "Freddy"}**
- Create another list variable called **tempList** within the Variables panel. Select the **Variable Type** same as that of **studentList**. Leave its **Default** column empty
- Insert an **Invoke Method** activity within the **Sequence** activity. Select the **TargetType** as **(null)**. Enter the variable **studentList** in the **TargetObject** field, and **Sort** in the **MethodName** field
- Insert a second **Invoke Method** activity below the previous Invoke Method activity. Select the **TargetType** as **(null)**. Enter the variable **studentList** in the **TargetObject** field, and **Reverse** in the **MethodName** field

- Insert an **Assign** activity below the second Invoke Method activity. Insert the variable **tempList** in the first input box, and in the adjacent box enter **studentList.GetRange(0,5)**
- Insert a Message Box activity below the Assign activity. Enter **String.Join(", ", tempList.ToArray)** in the text area
- Save and run the workflow
- Outcome
 - The names of five students are displayed in reverse alphabetical order in the message box

Slide 44

Practice Exercise





Build a workflow using the **Concat** and **Join** method that merges two lists containing the city names of the UK and Spain, sorts it, capitalizes the first letter of each item, and displays it in a message box.

- Create a list containing three cities of the UK in Uppercase
- Create another list containing three cities of Spain in Lowercase
- Merge both the lists together
- Sort the final list in alphabetical order from A to Z
- Capitalize only the first letter of all the items in the final list
- Display the final list in a message box in string format

Build a workflow using the **Concat** and **Join** method that merges two lists containing the city names of the UK and Spain, sorts it, capitalizes the first letter of each item, and displays it in a message box.

- Create a list containing three cities of the UK in Uppercase
- Create another list containing three cities of Spain in Lowercase
- Merge both the lists together
- Sort the final list in alphabetical order from A to Z
- Capitalize only the first letter of all the items in the final list
- Display the final list in a message box in string format


Process Overview

- START
- Create two List variables called **SpainCities** and **UKCities**
- For **SpainCities**, instantiate and populate the List from the Variables Panel, using the expression: **new List (of String) from {"madrid","valencia", "barcelona"}**
- For **UKCities**, instantiate the List from the Variables Panel using the expression: **new List (of String) from {"MANCHESTER","BRISTOL","GLASGOW"}**
- Merge the two List variables in a newly created List variable using **Enumerable.Concat** method inside an Assign activity
- Use the expression: **Enumerable.Concat(SpainCities.AsEnumerable, UKCities.AsEnumerable).ToList**


- **.AsEnumerable** converts the two Lists to Enumerable data type
 - **.ToList** converts the outcome to List
- Use an Invoke Method activity to sort the elements in the outcome List by specifying 'Sort' in the 'MethodName' field
- Use a For Each activity to iterate through each element in the **AllCities** variable and convert them to ProperCase using the StrConv method. Use the expression: **StrConv(item, VbStrConv.ProperCase)**
- Add the converted items to a newly created List variable **AllCitiesProperCase** using an Add to Collection activity
- Use a Message Box activity to display the converted values using the String.Join method. Use the expression: **String.Join(",",AllCitiesProperCase)**
- STOP

Slide 45

Summary



- 1 Data Manipulation and Its Importance
- 2 String Manipulation
- 3 DataTable Manipulation
- 4 Collection, Its Types, and Manipulation



To summarize, this lesson explained:

- Data Manipulation and Its Importance
- String Manipulation
- DataTable Manipulation
- Collection, Its Types, and Manipulation