GUVI

tech deserves you

RPA Design & Development V2.0

Student Manual

UiPath™ Learning Partner

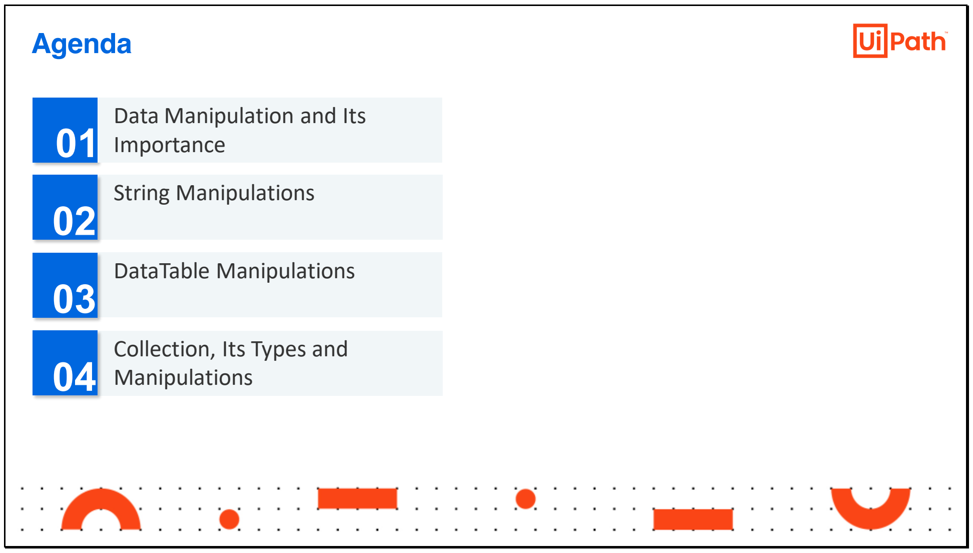
RPA Design and Development V2.0

UiPath

Welcome to 'RPA Design and Development Course'.

Lesson 6: Data Manipulation

The sixth lesson of this course is Data Manipulation.



Agenda

UiPath

01 Data Manipulation and Its Importance

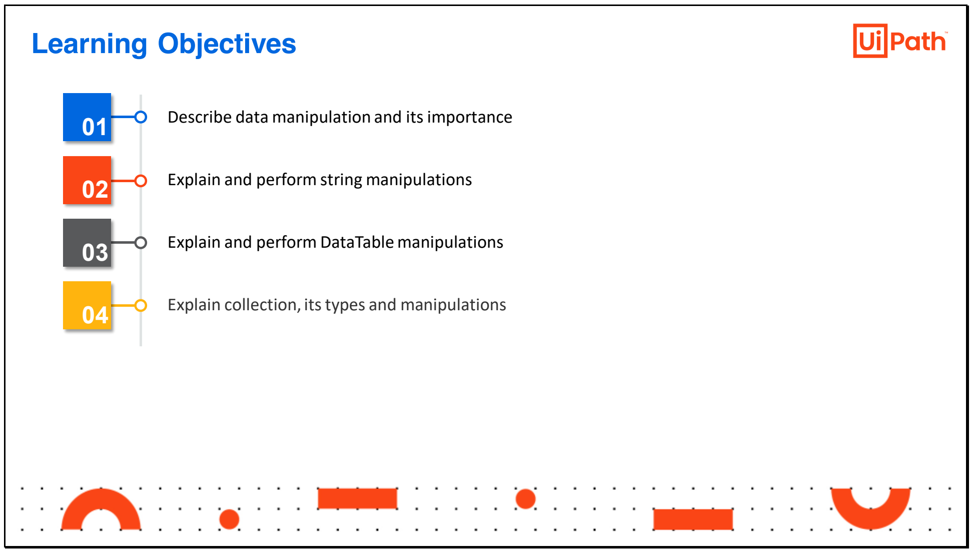
02 String Manipulations

03 DataTable Manipulations

04 Collection, Its Types and Manipulations

The agenda of this lesson is:

- Data Manipulation and Its mportance
- String Manipulations
- DataTable Manipulations
- Collection, Its Types and Manipulations



Learning Objectives

01 Describe data manipulation and its importance

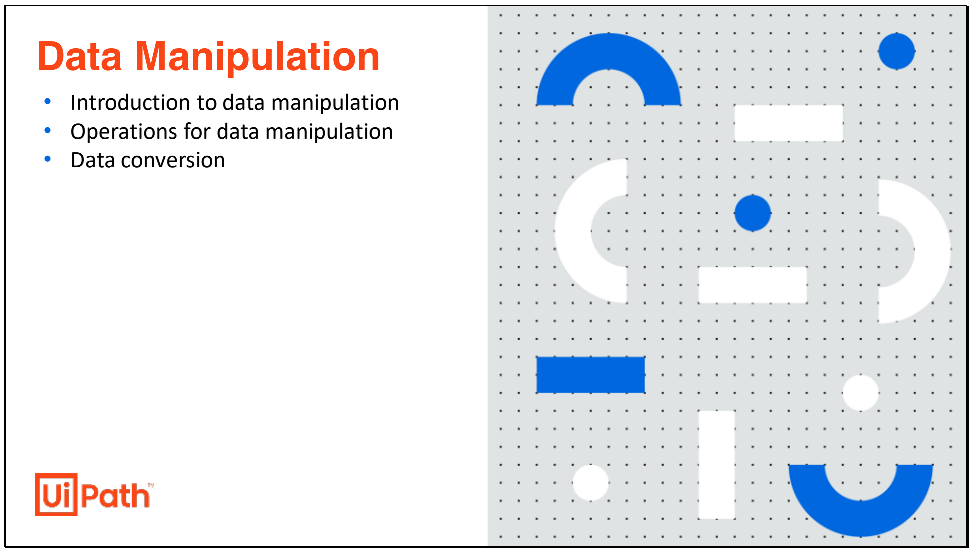
02 Explain and perform string manipulations

03 Explain and perform DataTable manipulations

04 Explain collection, its types and manipulations

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

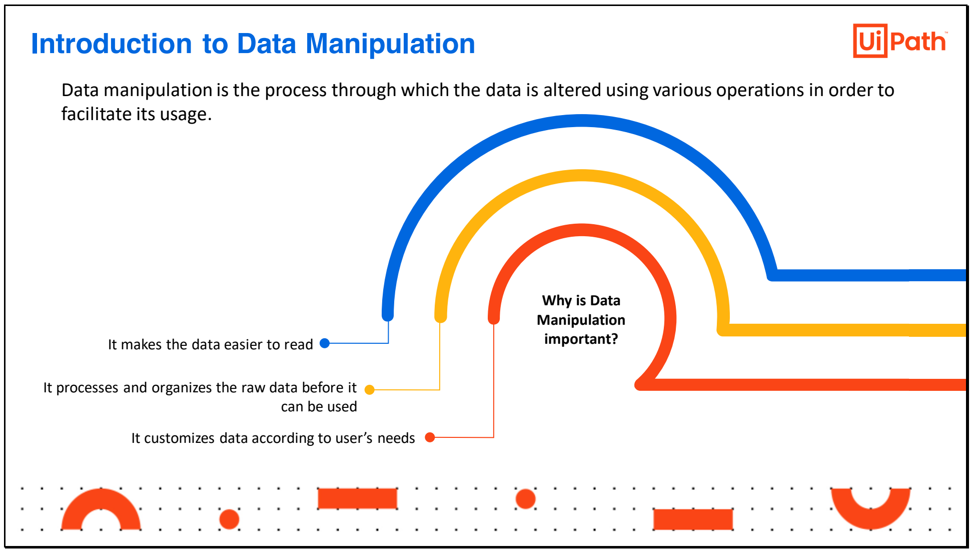
- Describe data manipulation and its importance
- Explain and perform string manipulations
- Explain and perform DataTable manipulations
- Explain collection, its types and manipulations



Data Manipulation

- Introduction to data manipulation
- Operations for data manipulation
- Data conversion

This section introduces data manipulation and data conversion.


Introduction to Data Manipulation

Data manipulation is the process through which the data is altered using various operations in order to facilitate its usage.

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 user's needs

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.

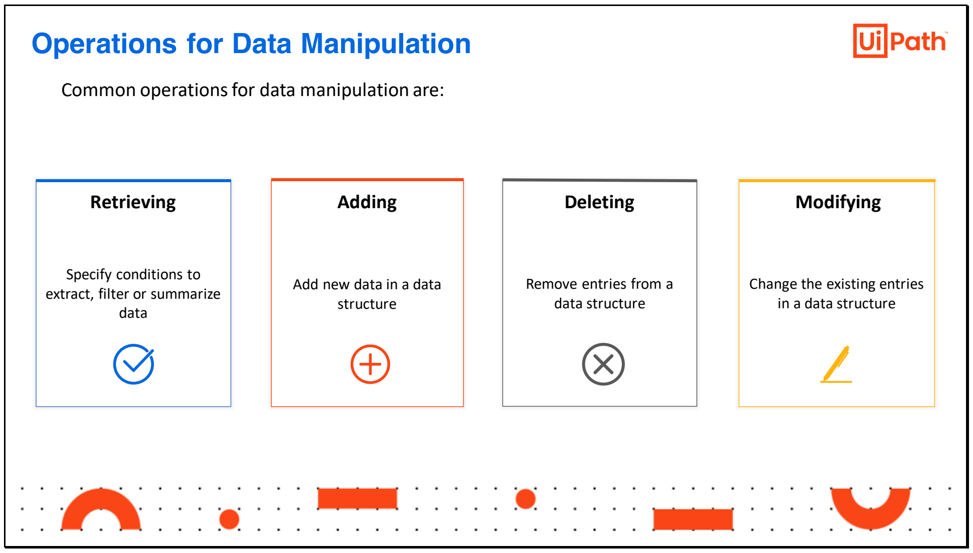
Why is Data Manipulation important?

- It makes the data easier to read
- It processes and organizes the raw data before it can be used
- It customizes data according to 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.



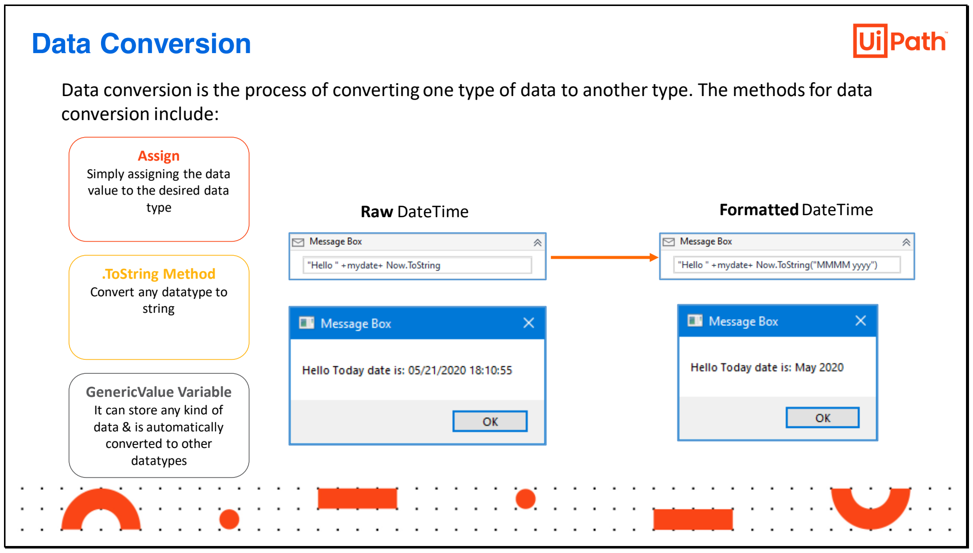
Operations for Data Manipulation

Common operations for data manipulation are:

Retrieving	Adding	Deleting	Modifying
Specify conditions to extract, filter or summarize data	Add new data in a data structure	Remove entries from a data structure	Change the existing entries in a data structure

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.



Data Conversion

Data conversion is the process of converting one type of data to another type. The methods for data conversion include:

Assign
Simply assigning the data value to the desired data type

.ToString Method
Convert any datatype to string

GenericValue Variable
It can store any kind of data & is automatically converted to other datatypes

Raw DateTime

```
"Hello " +mydate+ Now.ToString
```

```
Hello Today date is: 05/21/2020 18:10:55
```

Formatted DateTime

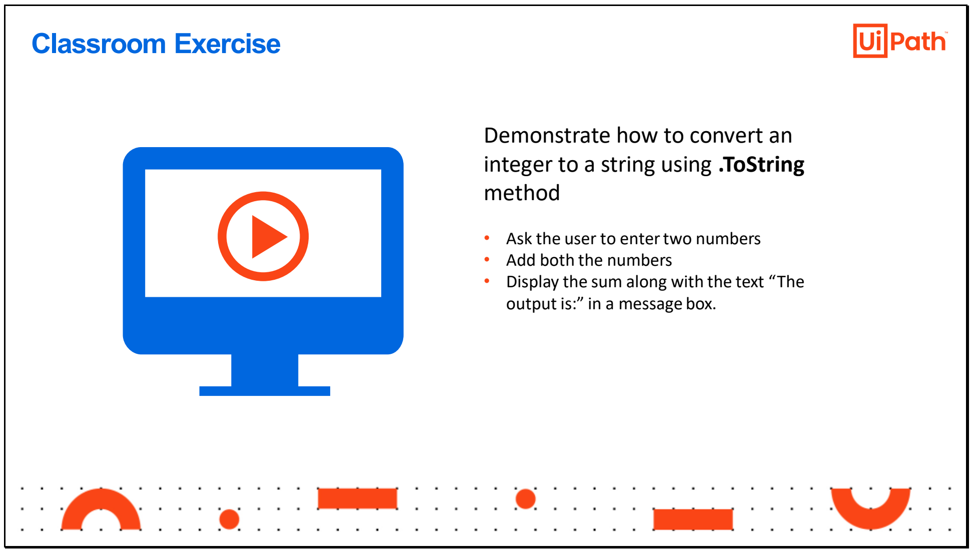
```
"Hello " +mydate+ Now.ToString("MMMM yyyy")
```

```
Hello Today date is: May 2020
```

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

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 string to a number.
- Convert any datatype to string using the .ToString method.
    - For example: The user can convert DateTime datatype to string datatype or number to string and vice versa.
    - An example of .ToString method is shown on the slide.
- Using GenericValue Variable: The GenericValue variable can store any kind of data, including text, numbers, dates, and arrays. GenericValue variables are automatically converted to other types, in order to perform certain actions. UiPath Studio has an automatic conversion mechanism of GenericValue variables, The first element in the expression is used as a guideline for what operation Studio performs.
    - For example: While adding two GenericValue variables, if the first one in the expression is defined as a String, the result is the concatenation of the two. If it is defined as an Integer, the result is their sum.



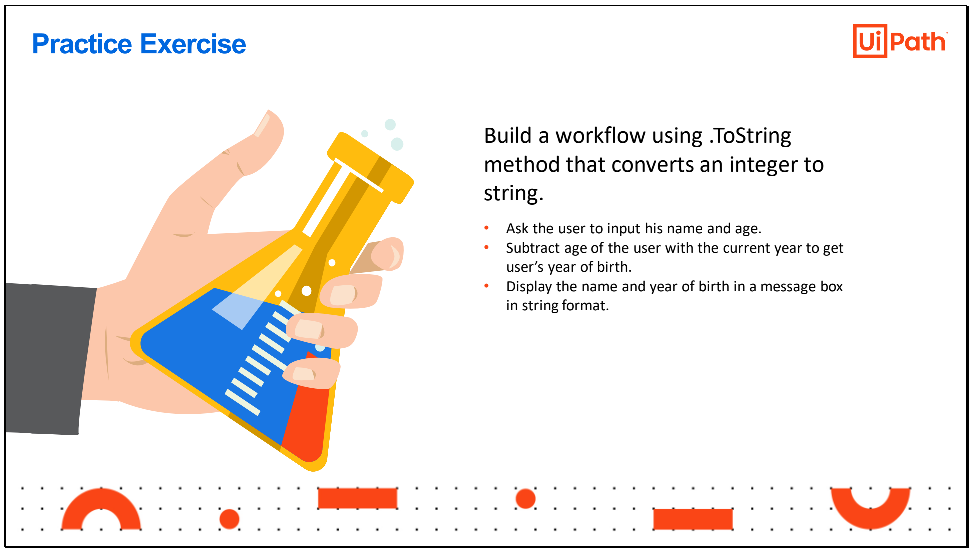
Classroom Exercise

Demonstrate how 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 how to use **.ToString** method to convert an integer to string. Add two numbers and display the result along with an 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 **firstNumber** for this activity using Variables panel. Set Variable Type as **Int32**. Enter this variable in **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 **secondNumber** for this activity using Variables panel. Set Variable Type as **Int32**. Enter this variable in **Result** property of this activity using its **Properties** panel.
- Insert **Assign** activity below the second **Input Dialog** activity. Press **Ctrl + K** and enter the text **totalSum** as variable in the *To* text box, and **firstNumber + secondNumber** in the adjacent box.
- Insert **Message Box** activity below the Assign activity. Enter text **"The output is: " + totalSum.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.



Practice Exercise

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

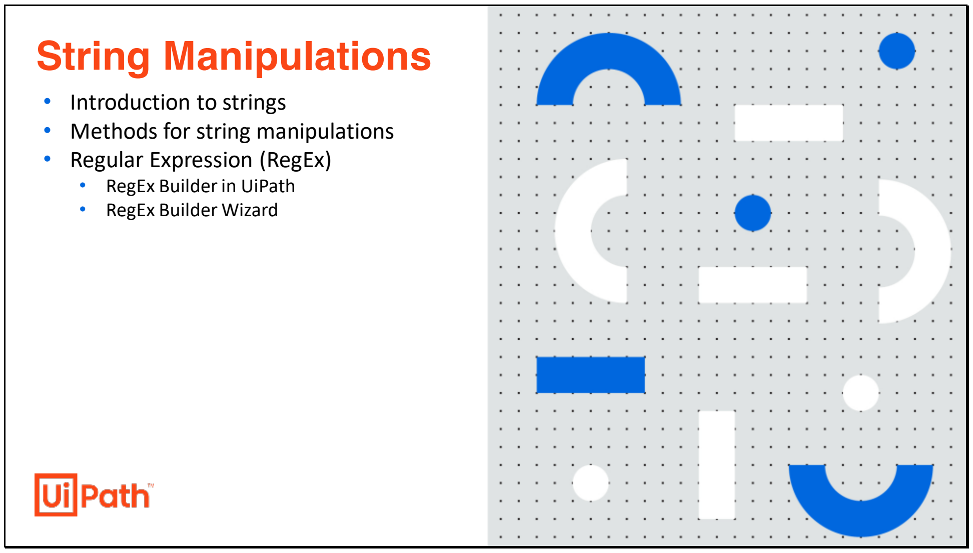
- Ask the user to input his name and age.
- Subtract age of the user with the current year to get user's year of birth.
- Display the name and year of birth in a message box in string format.

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

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

Algorithm

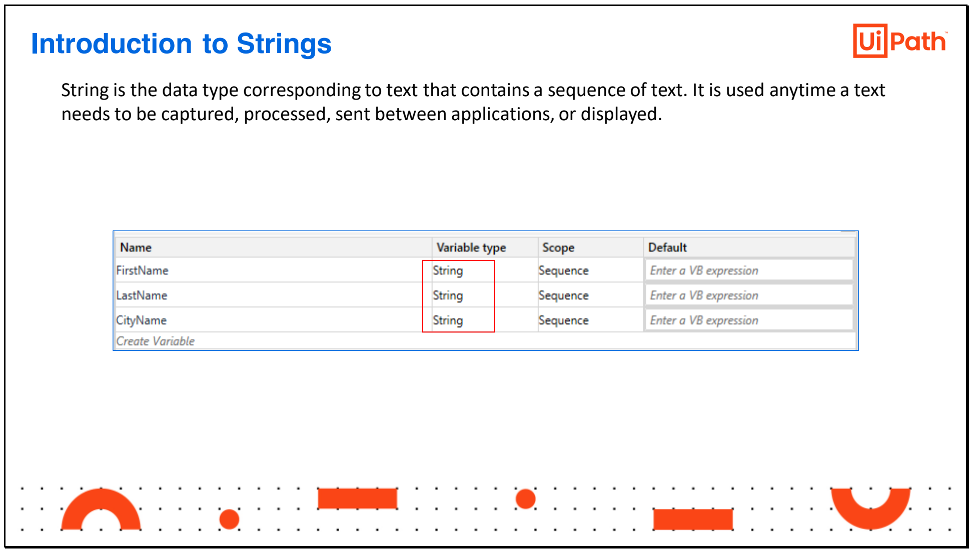
- START
- Use an Input Dialog activity to ask name and age from user
- Use an Assign activity and subtract age of the user from current year to get year of birth of the user.
- Use a Message Box activity to display: "Hello *User*, you were born in *Year*."
- Replace *User* and *Year* with name and the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



String Manipulations

- Introduction to strings
- Methods for string manipulations
- Regular Expression (RegEx)
  - RegEx Builder in UiPath
  - RegEx Builder Wizard

This section gives an overview of strings, its manipulations and RegEx.

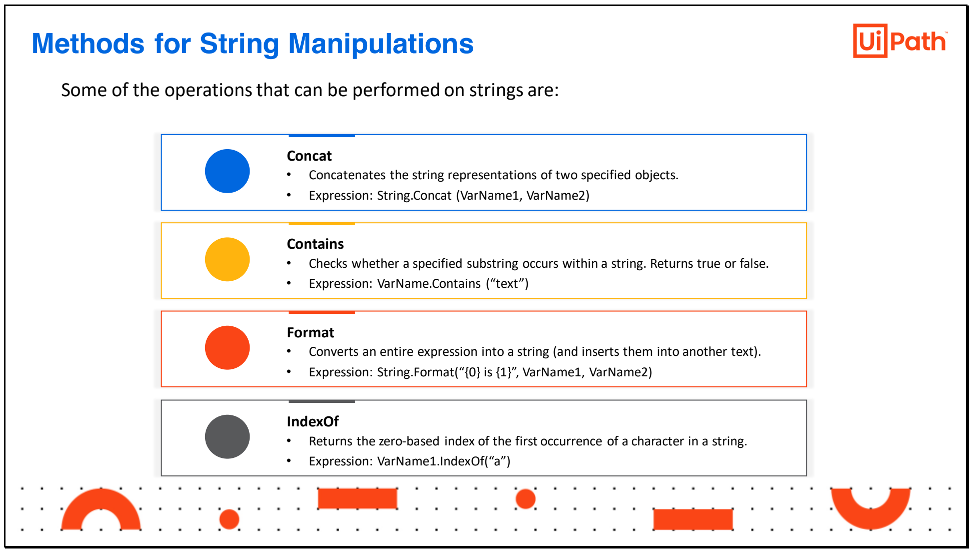


Introduction to Strings

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

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			

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 anytime a text needs to be captured, processed, sent between applications, or displayed.



Methods for String Manipulations

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)

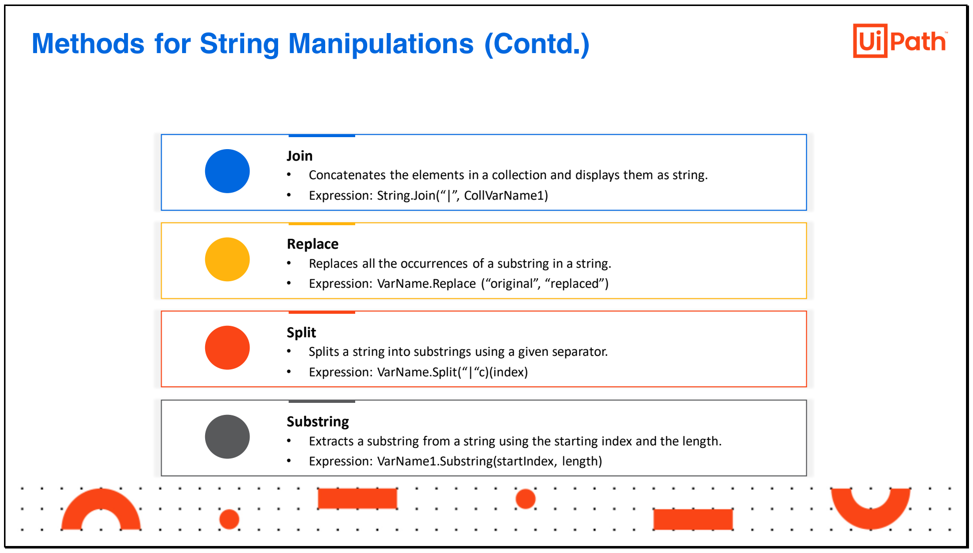
Contains
- Checks whether a specified substring occurs within a string. Returns true or false.
- Expression: VarName.Contains ("text")

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 Manipulations (Contd.)

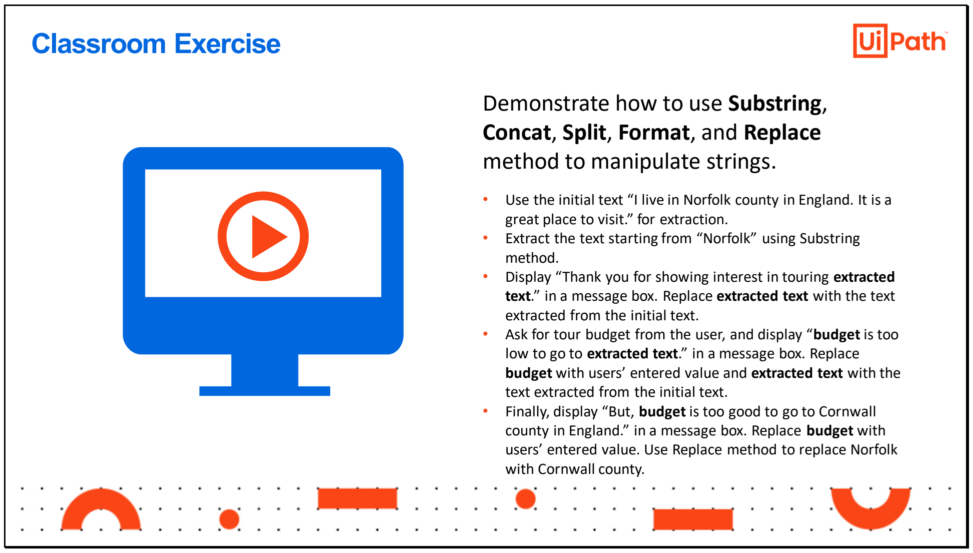
Join
- Concatenates the elements in a collection and displays them as string.
- Expression: String.Join("|", CollVarName1)

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 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 a certain criteria set by the user. This criterion could be a space, comma, or a 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)


Classroom Exercise

UiPath

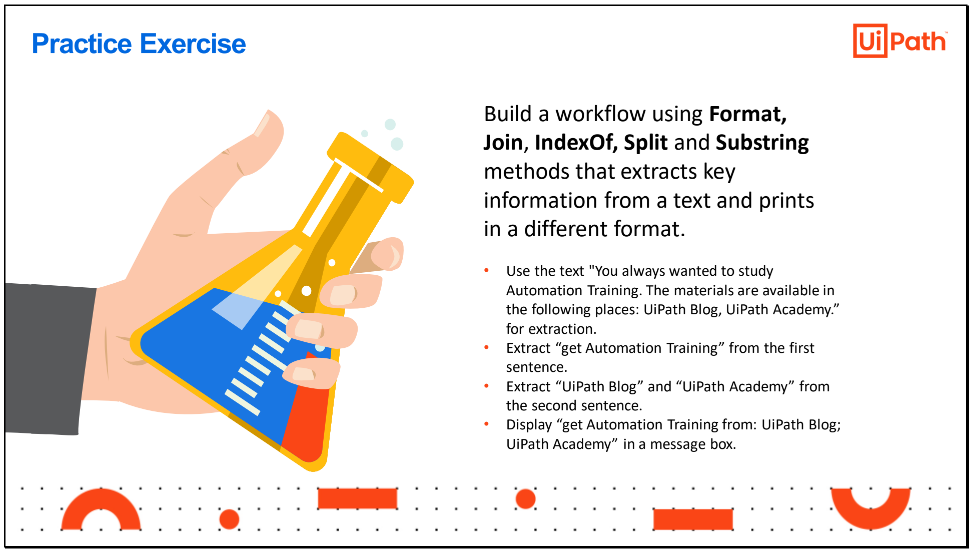
Demonstrate how to use **Substring**, **Concat**, **Split**, **Format**, and **Replace** method to manipulate strings.

- 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 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 how to use **Substring, Concat, Split, and Replace** method 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 user for tour budget, and suggest an alternative place to visit.

- Insert **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 Variables panel. Set Variable Type as **string**. Enter this variable in **Result** property of this activity using its **Properties** panel.
- Insert an **Assign** activity below 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 "get 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.

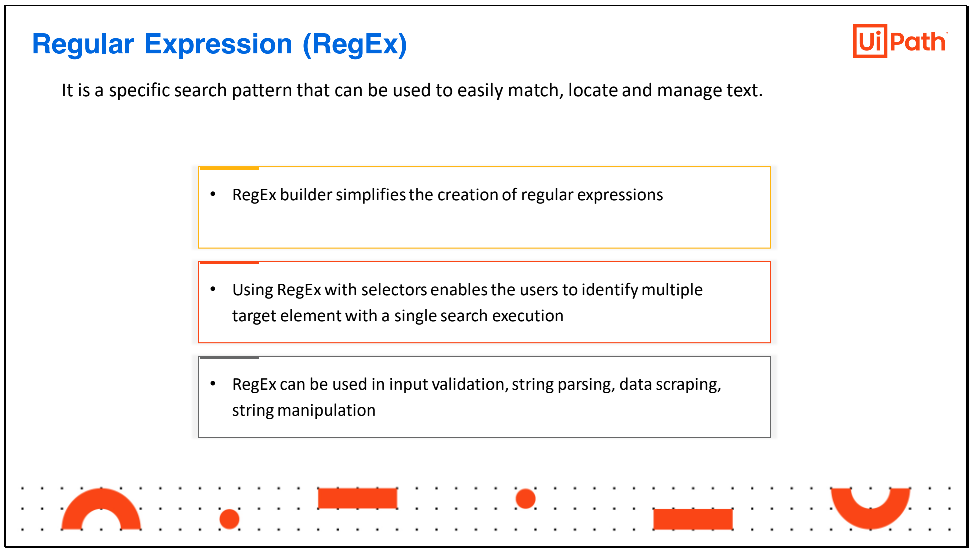
Build a workflow using Format, Join, IndexOf, Split and Substring methods that extracts key information from a text and prints in a different format.

- 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 "get 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.

Algorithm

- 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("get"))
  - **Split("."c).First.ToString** extracts the first sentence of the String and converts it to a String
  - **Substring(message.LastIndexOf("get"))** 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 Expression (RegEx)

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

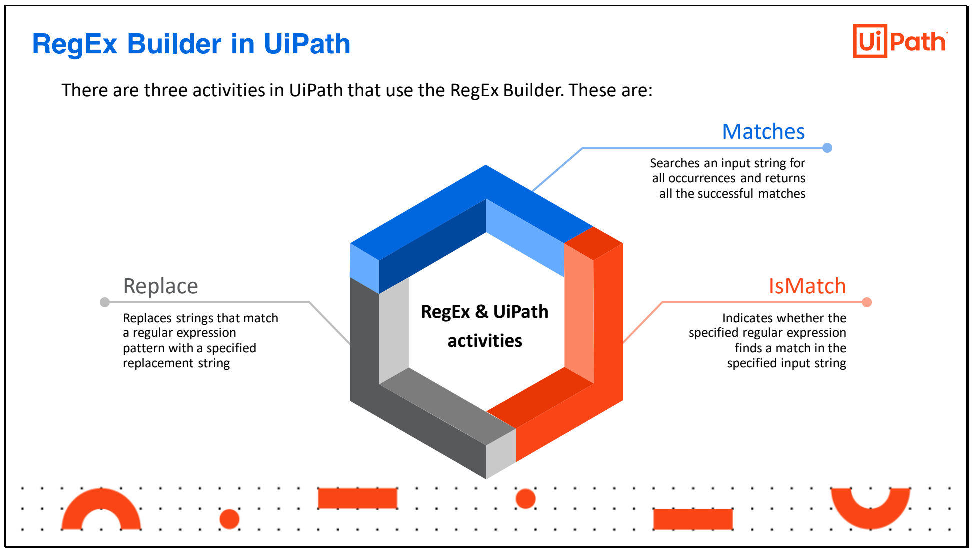
- RegEx builder simplifies the creation of regular expressions
- Using RegEx with selectors enables the users to identify multiple target element with a single search execution
- RegEx can be used in input validation, string parsing, data scraping, string manipulation

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

UiPath 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 element 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.



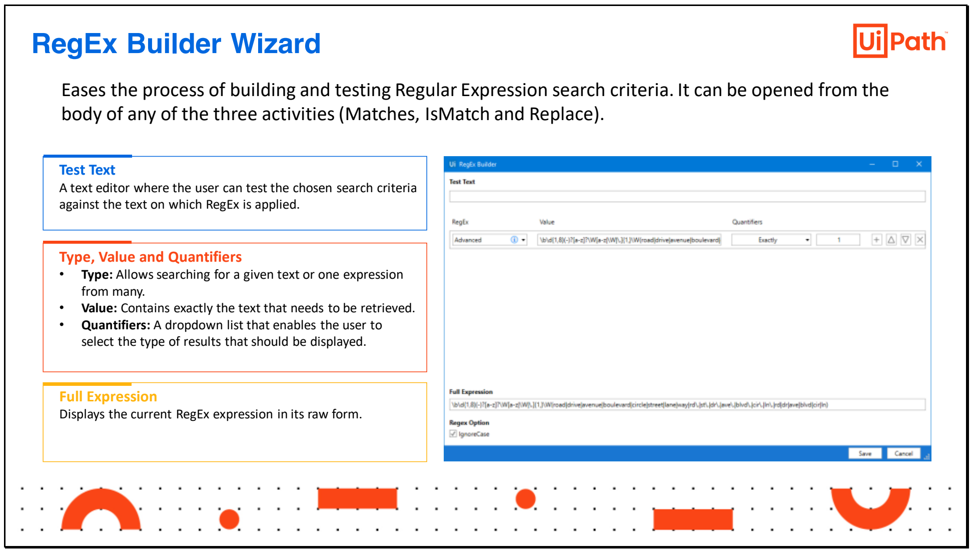
RegEx Builder in UiPath

There are three activities in UiPath that use the RegEx Builder. These are:

UiPath
Matches
Searches an input string for all occurrences and returns all the successful matches
Replace
Replaces strings that match a regular expression pattern with a specified replacement string
RegEx & UiPath activities
IsMatch
Indicates whether the specified regular expression finds a match in the specified input string

The RegEx Builder is integrated in three activities available in UiPath:

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



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, IsMatch 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.

RegEx: Advanced
Value: \b\d{1,8}(-)?[a-z]?\W[a-z|\W|\.]{1,}\W(road|drive|avenue|boulevard|
Quantifiers: Exactly 1

Full Expression: \b\d{1,8}(-)?[a-z]?\W[a-z|\W|\.]{1,}\W(road|drive|avenue|boulevard|circle|street|lane|way|rd\.|st\.|dr\.|ave\.|blvd\.|cir\.|ln\.|rd|dr|ave|blvd|cir|ln)

Regex Option: ✓ IgnoreCase

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, IsMatch 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.

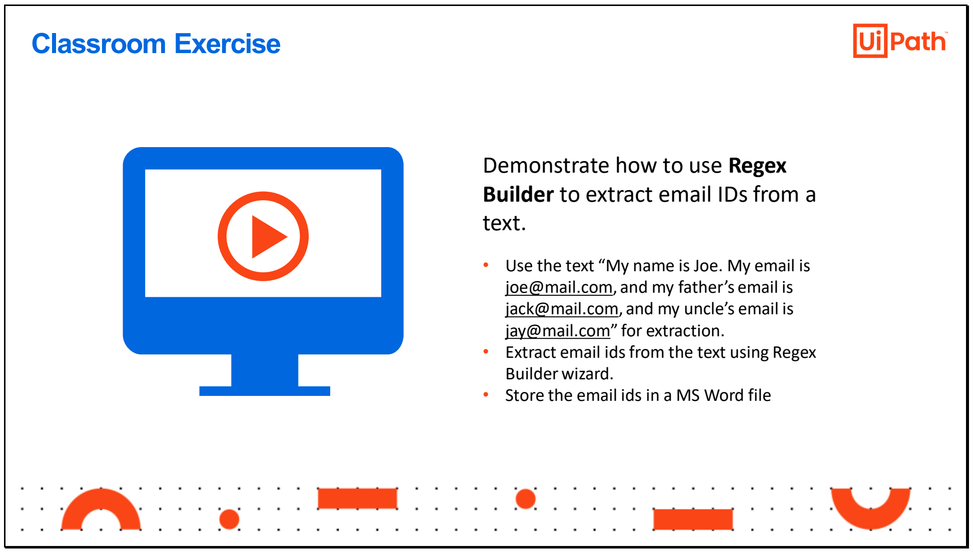
It simplifies building regular expressions by allowing the configuration of Type, Value and Quantifiers. Once a condition is finalized, another one can be added. If there are more than one conditions, the order in which they are applied can be configured as well.

- **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 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 Text Test 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 in the bottom of the wizard displays the current RegEx expression in its raw form.

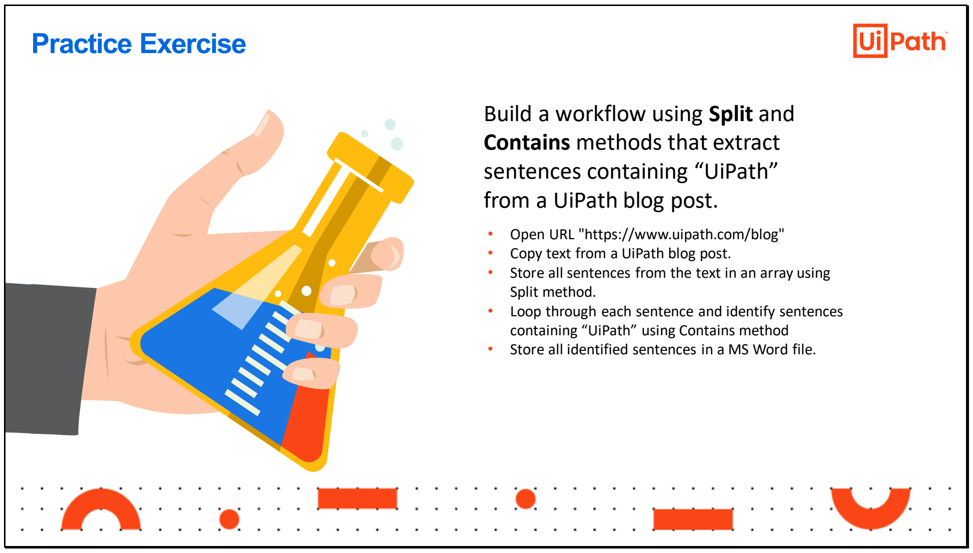

Classroom Exercise

Demonstrate how to use **Regex Builder** 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 a MS Word file

Demonstrate how to use Regex Builder 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 **Assign** activity in the designer panel. Press **Ctrl + K** and enter the text **message2** as 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 **Matches** activity below the Assign activity. Click on the **Configure Regular Expression** button within the Matches activity. In the **RegEx Builder** wizard, go to **RegEx** column and select **Email** from the drop down. In **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 **message2**. 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 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 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 "UiPath" from a UiPath blog post.

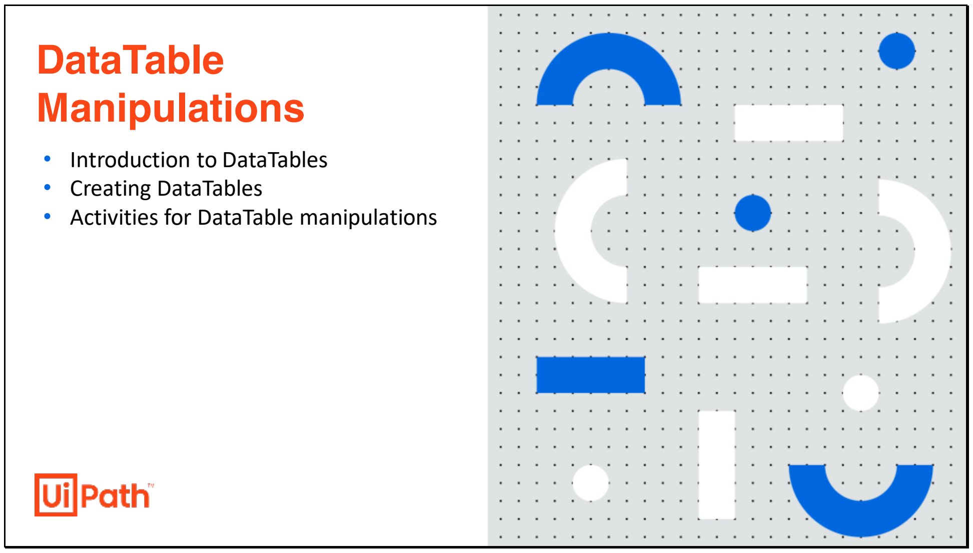
- Open URL "https://www.uipath.com/blog"
- Copy text from a UiPath blog post.
- Store all sentences from the text in an array using Split method.
- Loop through each sentence and identify sentences containing "UiPath" using Contains method
- Store all identified sentences in a MS Word file.

Build a workflow using Split and Contains methods that extracts sentences containing "UiPath" from a UiPath blog post.

- Open URL "https://www.uipath.com/blog"
- Copy text from a UiPath blog post.
- Store all sentences from the text in an array using Split method.
- Loop through each sentence and identify sentences containing "UiPath" using Contains method
- Store all identified sentences in a MS Word file.

Algorithm

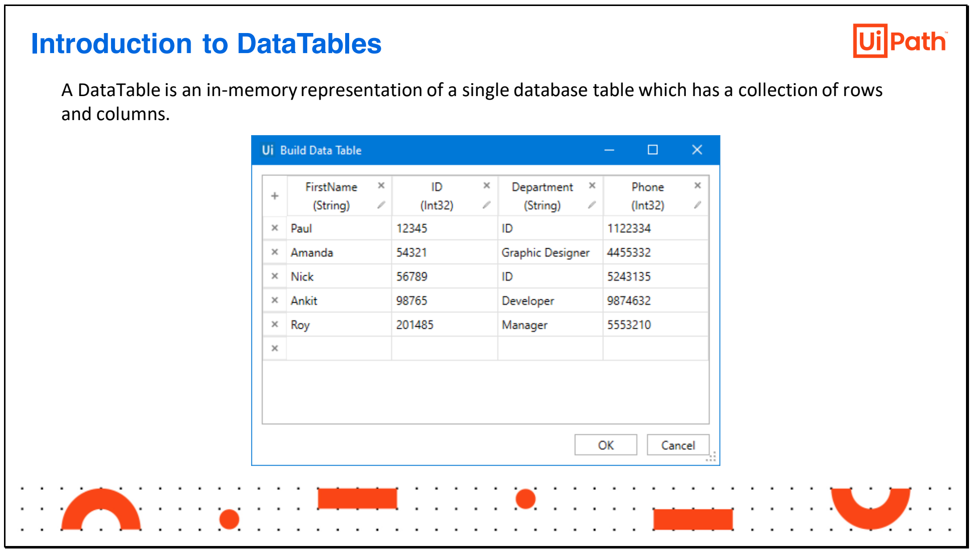
- START
- Use Open Browser activity to open URL "https://www.uipath.com/blog"
- Use Click Activity to open a blog post
- Use Get Text activity to store all text from a blog post of UiPath website in a string variable called **blogText**.
- Use **blogText.Split(".''c)** and store all sentences in an array called **blogSentence**
- Use For Each activity and iterate through each item in the array **blogSentence**
- Use If activity within For Each activity to identify sentence that contains the string "UiPath in it. Use **item.Contains("UiPath")** as condition.
- Use Type Into activity to store result in a MS Word file
- STOP



DataTable Manipulations

- Introduction to DataTables
- Creating DataTables
- Activities for DataTable manipulations

This section gives an overview of DataTable and its manipulations.



Introduction to DataTables

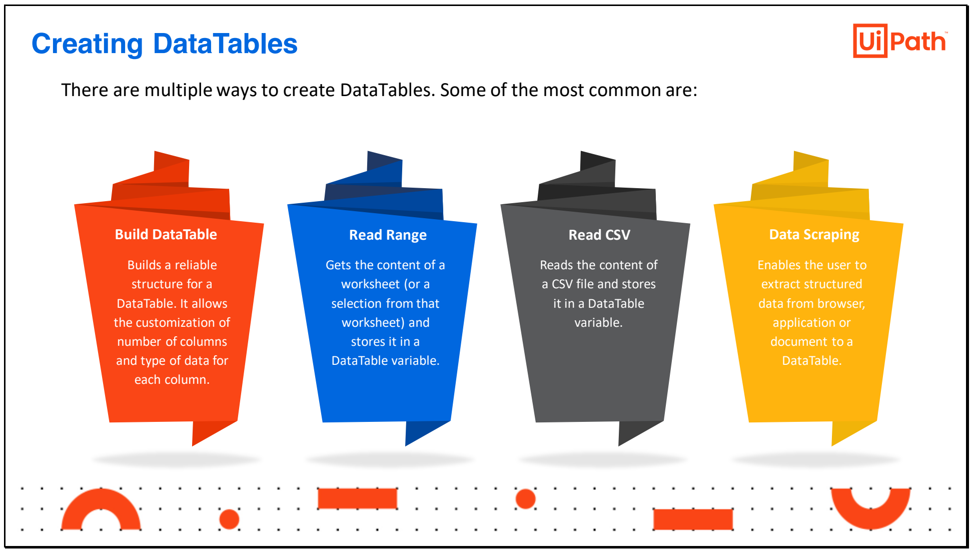
A DataTable is an in-memory representation of a single database table which has a collection of rows and columns.

Build Data Table

FirstName (String)	ID (Int32)	Department (String)	Phone (Int32)
Paul	12345	ID	1122334
Amanda	54321	Graphic Designer	4455332
Nick	56789	ID	5243135
Ankit	98765	Developer	9874632
Roy	201485	Manager	5553210

A DataTable is an in-memory representation of a single database table which 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 their unique column and row coordinates.

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



Creating DataTables

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

Build DataTable
Builds a reliable structure for a DataTable. It allows the customization of number of columns and type of data for each column.

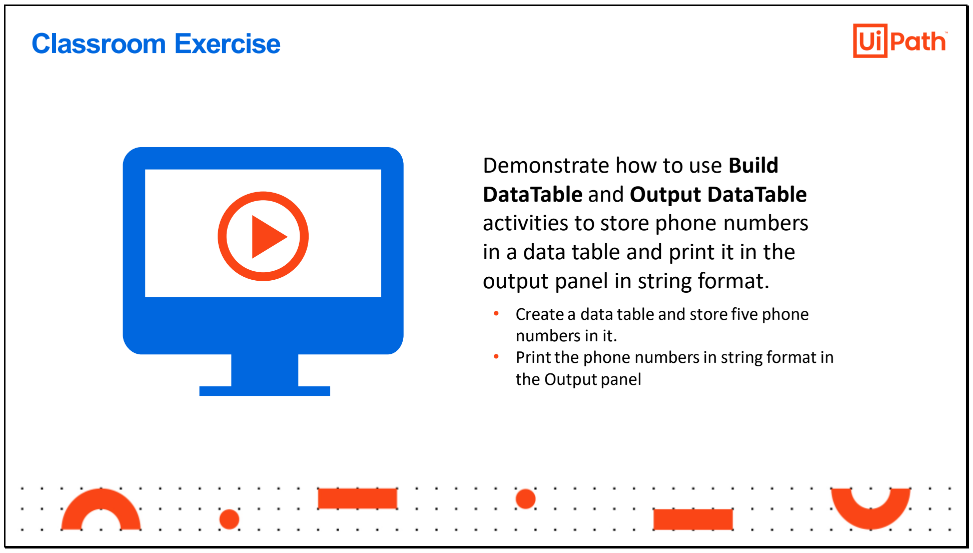
Read Range
Gets the content of a worksheet (or a selection from that worksheet) and stores it in a DataTable variable.

Read CSV
Reads the content of a CSV file and stores it in a DataTable variable.

Data Scraping
Enables the user to extract structured data from browser, application or document to a DataTable.

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 number of columns and 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 browser, application or document to a DataTable.



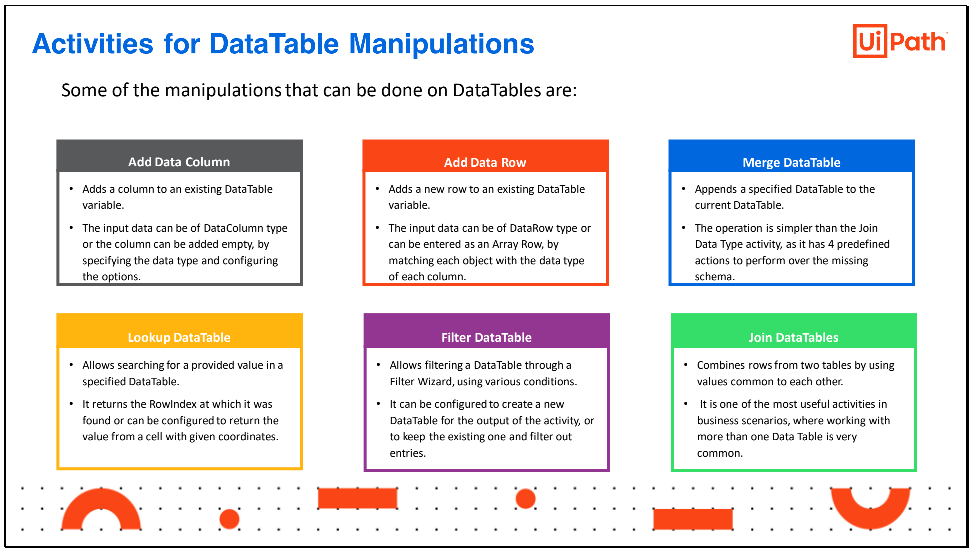
Classroom Exercise

Demonstrate how to use **Build DataTable** and **Output DataTable** 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 how to use 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 **Build Data Table** activity in the designer panel. Click the **Data Table** button within the activity. In the pop-up table, rename column header of 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 **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 **phoneNumbers**. In the Output property, press **Ctrl + K** and enter a new string variable called **stringPhoneNumbers**.
- Insert a **Write Line** activity below **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 Manipulations

UiPath

Some of the manipulations that can be done on DataTables 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 DataTable
- 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.

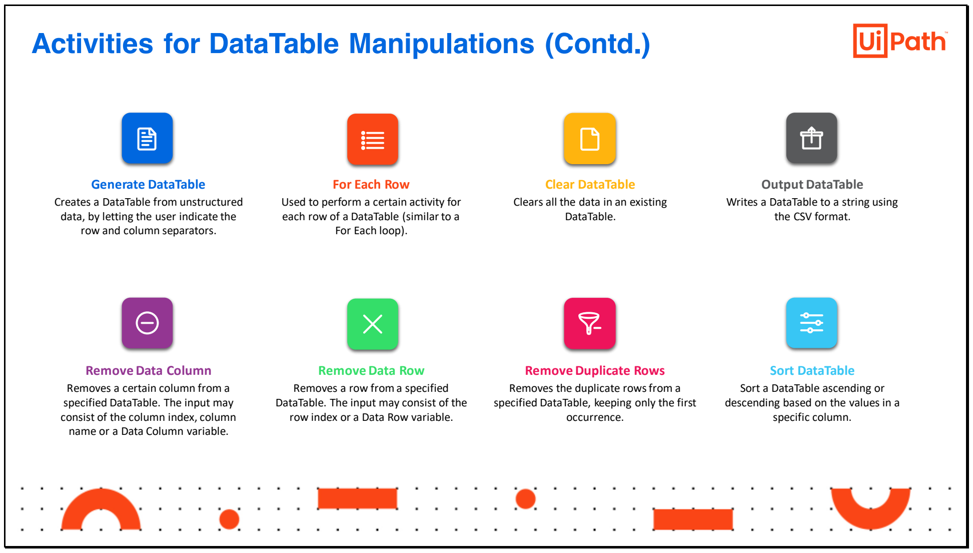
Lookup DataTable
- Allows searching for a provided value in a specified DataTable.
- It returns the RowIndex at which it was found or can be configured to return the value from a cell with given coordinates.

Filter DataTable
- 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 DataTables
- 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 DataTables 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
- **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.



Activities for DataTable Manipulations (Contd.)

Generate DataTable
Creates a DataTable from unstructured data, by letting the user indicate the row and column separators.

For Each Row
Used to perform a certain activity for each row of a DataTable (similar to a For Each loop).

Clear DataTable
Clears all the data in an existing DataTable.

Output DataTable
Writes a DataTable to a string using the CSV format.

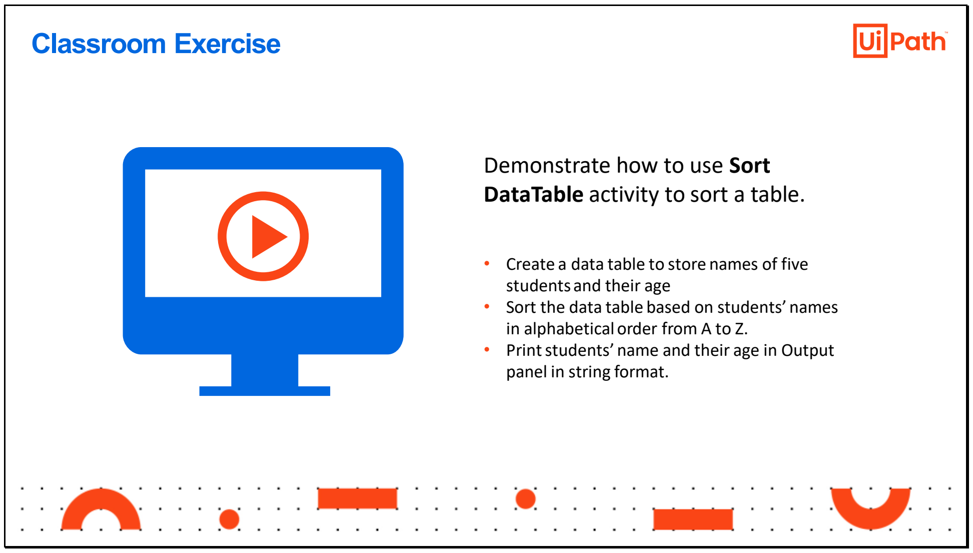
Remove Data Column
Removes a certain column from a specified DataTable. The input may consist of the column index, column name or a Data Column variable.

Remove Data Row
Removes a row from a specified DataTable. The input may consist of the row index or a Data Row variable.

Remove Duplicate Rows
Removes the duplicate rows from a specified DataTable, keeping only the first occurrence.

Sort DataTable
Sort a DataTable ascending or descending based on the values in a specific column.

- **Generate DataTable**: 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**: 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 column level (for example, verification of the relation between two values stored in two columns).
- **Clear DataTable**: 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.
- **Output DataTable**: 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.
- **Remove Data:** It is a group of two activities used to remove rows or columns from an existing DataTable. Rows can be identified by their index number or by inputting the DataRow as object, and the Columns can be additionally identified by their names.
- **Remove Duplicate Rows**: It is used to remove the duplicate rows, while keeping only the first occurrence.
- **Sort DataTable**: 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.


Classroom Exercise

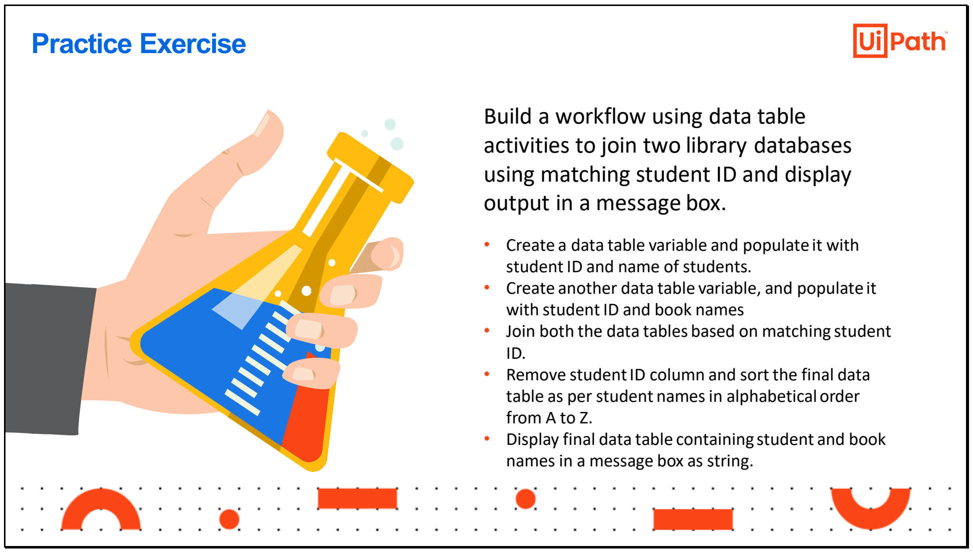
Demonstrate how to use **Sort DataTable** activity to sort a table.

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

Demonstrate how to use 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 age in the Output panel in string format.

- Insert **Build Data Table** activity in the designer panel. Click on **DataTable** button within 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 **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 **sortedTable**. In the **Index** property, enter value **0**. In the **Order** property, choose **Ascending**.
- Now, insert **Output Data Table** activity below the Sort Data Table activity. Go to its **Properties** panel, and enter **sortedTable** in the Input property. In the **Output** property, press **Ctrl + K** on your keyboard, and enter a new variable called **sortedByNames**.
- Insert **Write Line** activity below the Output Data Table activity. Enter the variable **sortedByNames**.
- 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 age.



Practice Exercise

Build a workflow using data table 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 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 final data table containing student and book names in a message box as string.

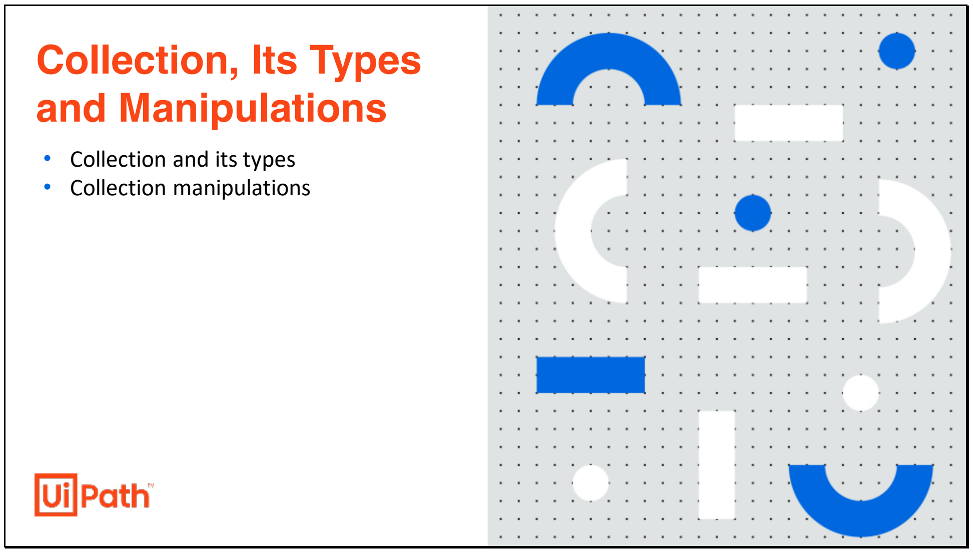
Build a workflow using data table activities to join two library databases using matching student ID and display output in a message box.

- Create a data table variable and populate it with student ID and 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 final data table containing student and book names in a message box as string.

Algorithm

- START
- Use two Build Data Table activities to create two tables. Store them in two DataTable variables called **users** and **overdueBooks**.
    - **users** variable contain ID of students as integer and name of user as string
    - **overdueBooks** variable contain ID of students as integer 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 **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 a Output Data Table activity to print the content of the data table to a String variable
- Use Message Box activity to display the output.
- STOP

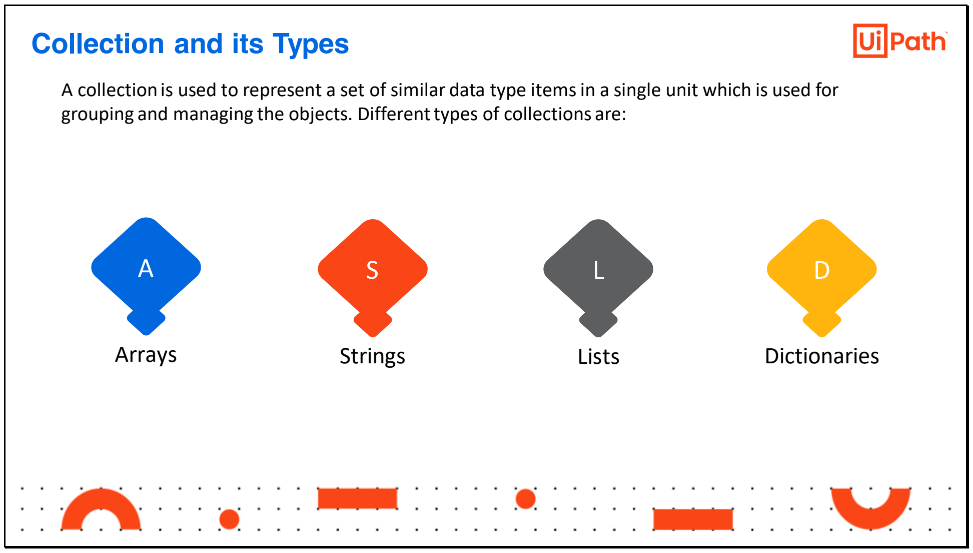


Collection, Its Types and Manipulations

- Collection and its types
- Collection manipulations

UiPath

This section explains collections and their manipulations in detail.


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:

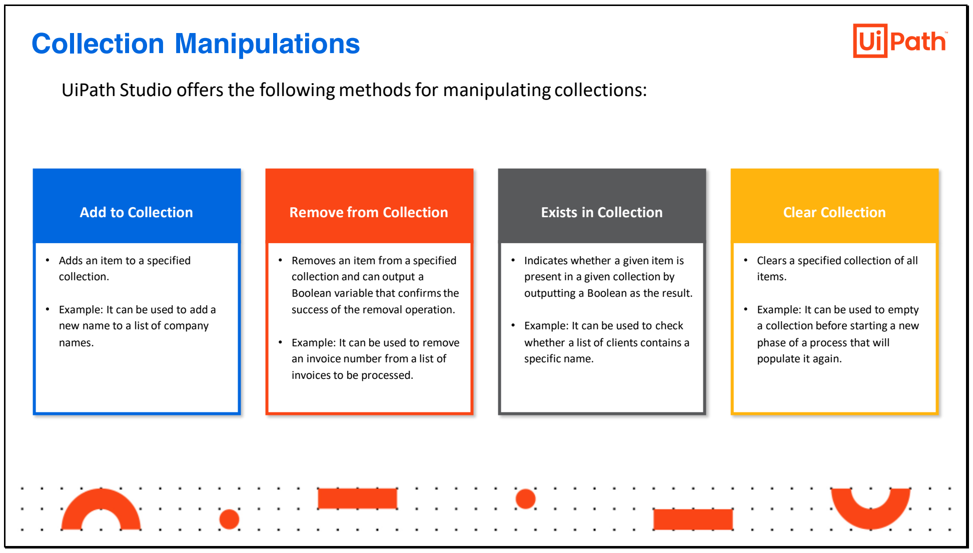
A
Arrays
S
Strings
L
Lists
D
Dictionaries

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
- Strings
- Lists
- Dictionaries

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



Collection Manipulations

UiPath Studio offers the following methods for manipulating collections:

Add to Collection
- Adds an item to a specified collection.
- Example: It can be used to add a new name to a list of company names.

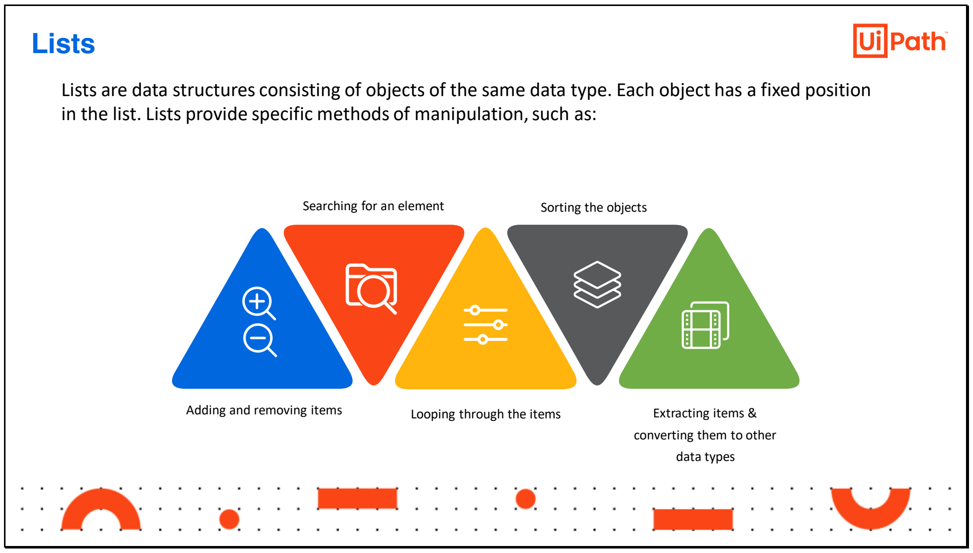
Remove from Collection
- Removes an item from a specified collection and can output a Boolean variable that confirms the success of the removal operation.
- Example: It can be used to remove an invoice number from a list of invoices to be processed.

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.

UiPath 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.



Lists

UiPath

Lists are data structures consisting of objects of the same data type. Each object has a fixed position in the list. Lists provide specific methods of manipulation, such as:

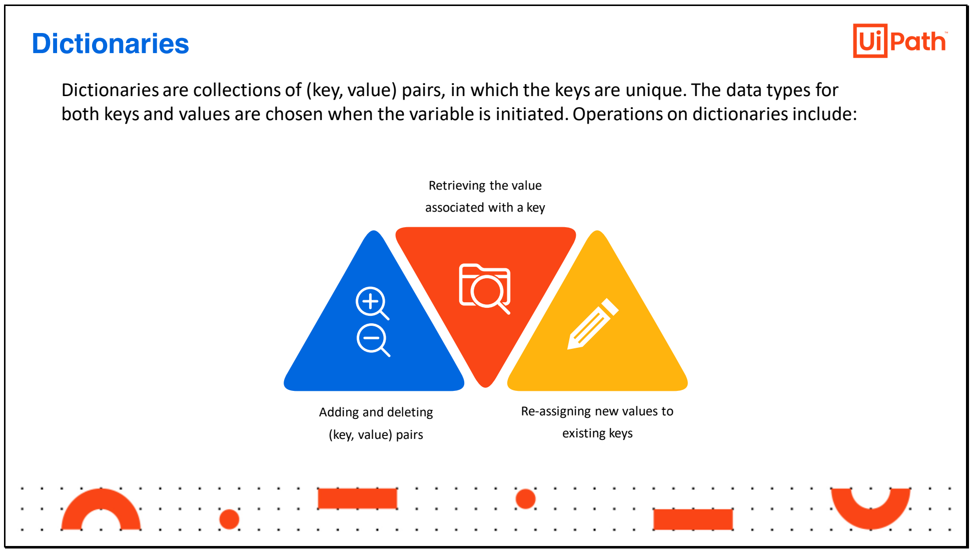
Searching for an element
Sorting the objects
Adding and removing items
Looping through the items
Extracting items & converting them to other data types

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



Dictionaries

Dictionaries are collections of (key, value) pairs, in which the keys are unique. The data types for both keys and values are chosen when the variable is initiated. Operations on dictionaries include:

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

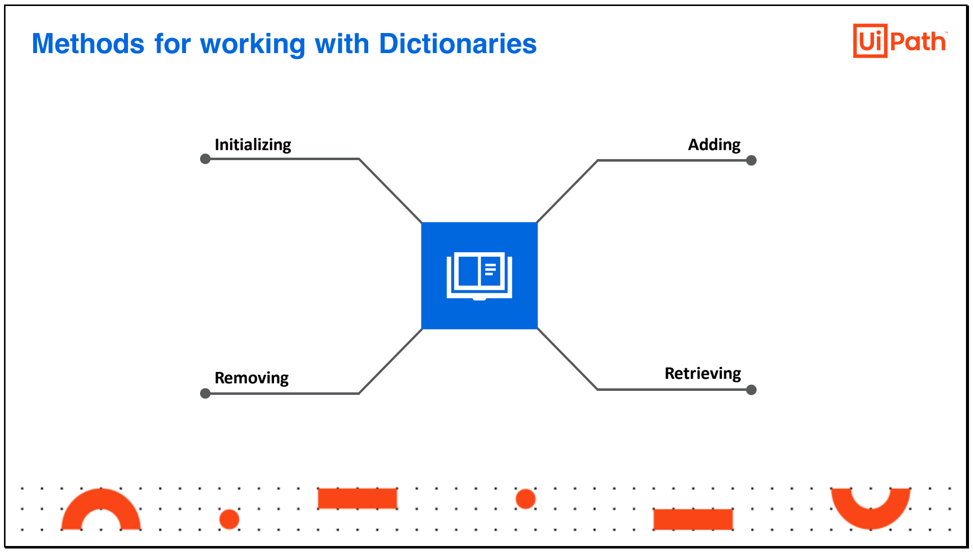
A dictionary is a collection of words and their meanings/definitions. Dictionaries consist of (key, value) pairs, in which the keys are unique. Here, key is equivalent to words and value is equivalent to meanings/definitions.

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 instantiated. Data types in dictionaries can be any of the supported variables.

The operations that are most often associated with dictionaries are:

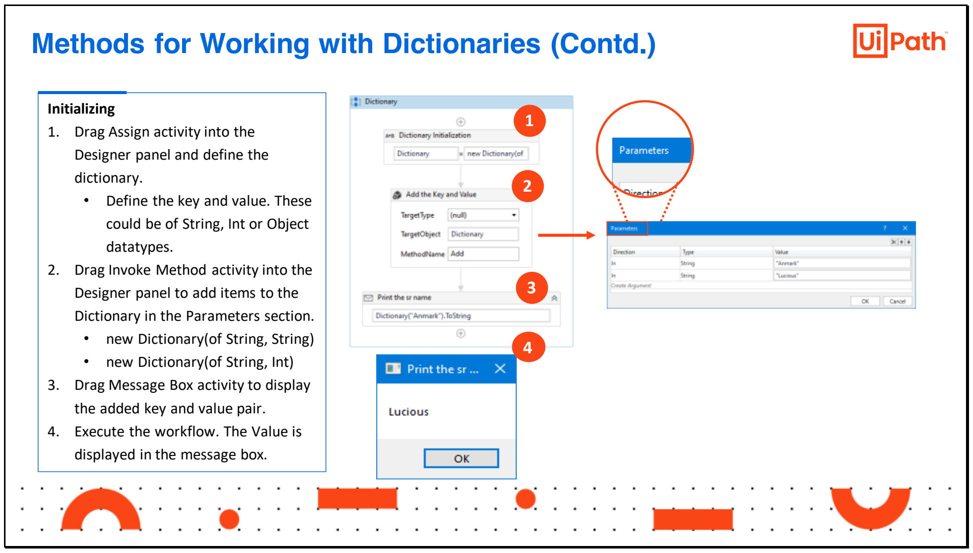
- Adding and deleting (key, value) pairs
- Retrieving the value associated with a key
- Re-assigning new values to existing keys


Methods for working with Dictionaries

There are several methods for working with the dictionaries:

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

These are discussed in detail in the subsequent slides.



Methods for Working with Dictionaries (Contd.)

Initializing

1. Drag Assign activity into the Designer panel and define the dictionary.
   - Define the key and value. These could be of String, Int or Object datatypes.
2. Drag Invoke Method activity into the Designer panel to add items to the Dictionary in the Parameters section.
   - new Dictionary(of String, String)
   - new Dictionary(of String, Int)
3. Drag Message Box activity to display the added key and value pair.
4. Execute the workflow. The Value is displayed in the message box.

Dictionary

1 Dictionary Initialization: Dictionary = new Dictionary(of

2 Add the Key and Value

TargetType	(null)
TargetObject	Dictionary
MethodName	Add

Parameters

Direction	Type	Value
In	String	"Anmark"
In	String	"Lucious"

3 Print the sr name: Dictionary("Anmark").ToString

4 Print the sr ...

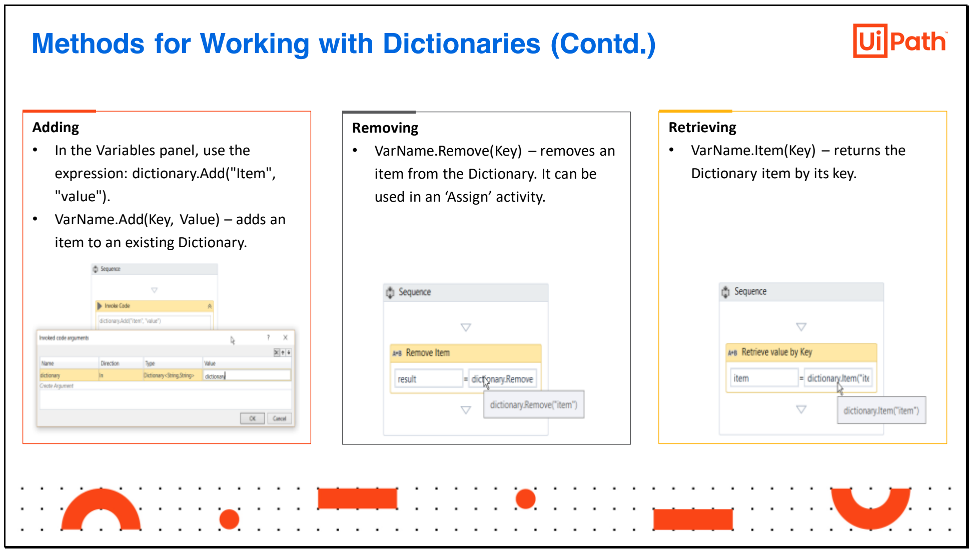
Lucious

OK

Initializing:

- Drag Assign activity into the Designer panel and define the dictionary.
  - Define the key and value. These could be of String, Int or Object datatypes.
- Drag Invoke Method activity into the Designer panel to add items to the Dictionary. In the Properties panel of the Invoke Method activity, go to the Parameters section and define the key and value pair.
  - new Dictionary(of String, String): Here, the datatype for key and value is String.
  - new Dictionary(of String, Int) : Here, the datatype is String for key and Int for value.
- Drag Message Box activity to display the added key and value pair.
- Execute the workflow.

In the example on the slide, the key is 'Ainmark' and the value is 'Lucious'. When the workflow is executed, it displays the sr name 'Lucious' in the message box activity.



Methods for Working with Dictionaries (Contd.)

Adding
- In the Variables panel, use the expression: dictionary.Add("Item", "value").
- VarName.Add(Key, Value) – adds an item to an existing Dictionary.

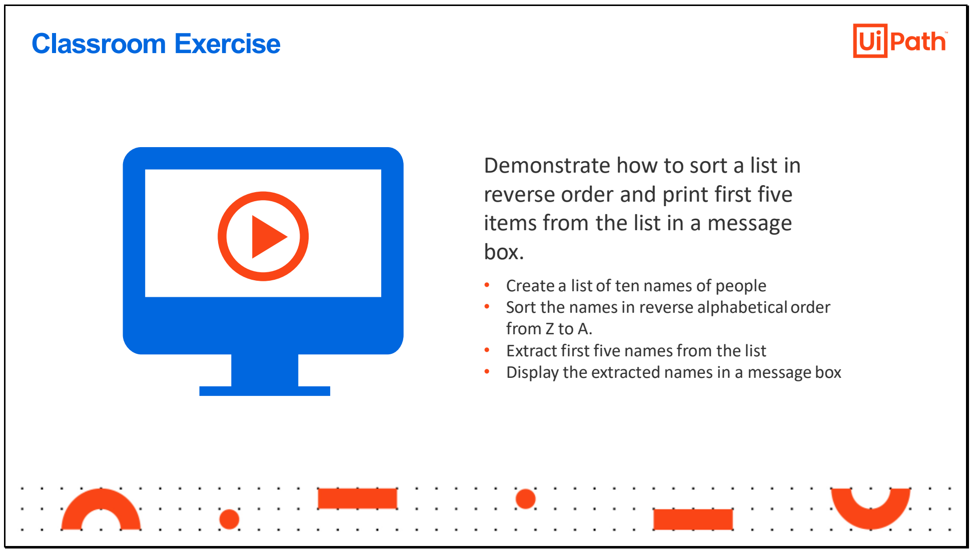
Removing
- VarName.Remove(Key) – removes an item from the Dictionary. It can be used in an 'Assign' activity.

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

Adding:
- In the Variables panel, use the expression: dictionary.Add("Item", "value").
- VarName.Add(Key, Value): Adds an item to an existing dictionary. Add does not return a value, so the Invoke Code activity is used.

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.



Classroom Exercise

Demonstrate how to sort a list in reverse order and print 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 first five names from the list
- Display the extracted names in a message box

Demonstrate how to sort a list in reverse order and print 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 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 **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 **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 **Variable Type** same as that of **studentList**. Leave its **Default** column empty.
- Insert **Invoke Method** activity within the **Sequence** activity. Select **TargetType** as **(null)**. Enter the variable **studentList** in **TargetObject** field, and **Sort** in **MethodName** field.
- Insert second **Invoke Method** activity below previous Invoke Method activity. Select **TargetType** as **(null)**. Enter the variable **studentList** in **TargetObject** field, and **Reverse** in **MethodName** field.
- Insert **Assign** activity below 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 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.

Practice Exercise

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

- Create a list containing three UK cities in capital letters
- Create another list containing three Spain cities in small letters
- 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.

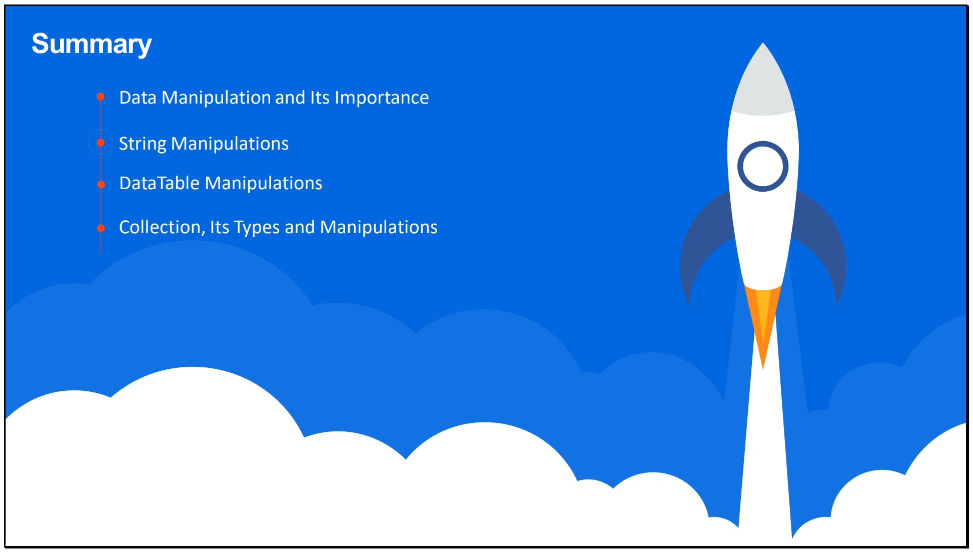
Build a workflow using **Concat** and **Join** method that merges two lists containing UK and Spain city names, sorts it, capitalizes first letter of each item, and displays it in a message box.

- Create a list containing three UK cities in all caps
- Create another list containing three Spain cities in small letters
- 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.

Algorithm

- 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 elements in the **AllCities** variable and
- Convert them to ProperCase using the StrConv method. User the expression: **StrConv(item, VbStrConv.ProperCase)**
- Add the converted items to a newly created List variable **AllCitiesProperCase** using 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



Summary

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

To summarize, this lesson explained:

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