ROBOTIC PROCESS AUTOMATION DESIGN & DEVELOPMENT (18CS745)

MODULE 3

SEQUENCE, FLOWCHART, AND CONTROL FLOW

Referred Textbook: Alok Mani Tripathi, Learning Robotic Process Automation, Publisher: Packt Publishing Release Date: March 2018 ISBN: 9787788470940

MODULE 3

SEQUENCE, FLOWCHART, AND CONTROL FLOW

Sequencing the workflow

UiPath provides four types of projects:

- Sequences
- Flowcharts
- User Events
- State Machines

A Flowchart and Sequence are mainly used for simple automation. User Events are beneficial for implementing front office robots, while State Machines are used for dealing with complex business processes.

What is a Sequence?

A Sequence is a group of logical steps. Each step represents an action or a piece of work. A Sequence is used for processes that happen in linear succession, that is, one after the other. Among the three types of projects in UiPath, Sequences are the smallest.

Activities:

In UiPath Studio, an activity represents the unit of an action. Each activity performs some action. When these activities combine together, it becomes a process. Every activity resides on the Activities panel of the main Designer panel.

We can search for a particular activity and use it in your project. For example, when we search for browser, all the browser activities will appear in the Activities panel.

Workflow:

Suppose you have a big project that consists of hundreds of activities. To build such a big project, a developer will simply divide it into smaller modules and extract it as a workflow. Now, each workflow can be tested separately. Thus, it is very easy to find bugs and errors. Creating different workflows and combining them into a logical Sequence will enhance our code quality, maintainability, reliability, and readability.

To create a workflow in UiPath Studio , we should Right-click on the main Designer panel and choose Extract as Workflow.

A window will pop up asking for the name. Give it a meaningful name and click on Create. This will be the name of your workflow

What Flowcharts are and when to use them

A Flowchart is generally used for complex business processes. It provides decision-making facilities and can be used for both small and large projects.

A Flowchart provides multiple branching logical operators to make decisions. A Flowchart is able to run in reverse. Also, it can be used inside Sequences. A Flowchart facilitates reusability for distinct projects. Once we create it to use in a project, it can be used for a different but similar project. A Flowchart's branches are set to true/false by default. However, its names can be manually changed from the Properties panel.

For example, enter two numbers and check whether their sum is less than 20.
Perform the following steps:

1. First, add a **Flowchart** from the Activities panel into the Designer panel.
2. Add a **Sequence** activity within the Flowchart.
3. Take two **Input dialog** activities (for entering the numbers to be added) inside the Sequence activity.
4. Create the variables x and y to save the values.
5. Next, add a **Message box** activity to perform a mathematical operation. In our case, the sum of the two numbers is less than 20:

$$x + y < 20$$

6. Now, add a **Flow Decision** activity to check the mathematical operation.
7. If true, the Flow Decision will flow toward the true branch. Otherwise, it will flow towards the false branch.

Control flow, various types of loops, and decision making:

Control flow refers to the order or the particular manner in which actions are performed in an automation. UiPath provides numerous activities for performing the decision-making process. These activities, present in the Activities panel, are put into the workflow either using the double-click method or the drag and drop method.

Different types of control flow activities are as follows:

- The Assign activity
- The Delay activity
- The Break activity
- The While activity
- The Do While activity
- The For each activity
- The If activity
- The Switch activity

The Assign activity:

- The Assign activity is used to designate a value to the variable.
- The Assign activity can be used for different purposes, such as incrementing the value of a variable in a loop, or using the results of a sum, difference, multiplication, or division of variables and assigning it to another variable.

The Delay activity:

- The Delay activity, as the name suggests, is used to delay or slow down an automation by pausing it for a defined period of time.
- The workflow continues after the specified period of time. It is in the hh:mm:ss format.
- This activity plays a significant role when we need a waiting period during automation, perhaps say, a waiting period required for a particular application to open.

Example:

To better understand how the Delay activity works, let us see an example of an automation that writes two messages to the Output panel with a delay of 50 seconds.

Perform the following steps:

1. First, create a new **Flowchart**.

2. Add a **Write line** activity from the **Activities** panel and connect it to the **Start** node.

3. Select the **Write line** activity. Now, type the following text into the **Text** box: “What is Your Name?”

4. Next, add a **Delay** activity and connect it to the **Write line** activity.

5. Select the **Delay** activity and go to the **Properties** panel. In the **Duration** field, set 00:00:50. This is a 50-second delay between the two logged messages.

6. Take another **Write line** activity and connect it to the **Delay** activity. In the Text field, write

   “My name is Andrew.”

7. After clicking on the Run button, the Output panel shows the message that delays it by 50 seconds.

The Break activity:

- The Break activity is used to break/stop the loop at a particular point, and then continue to the next activity according to the requirement.
- It cannot be used for any other activity apart from the For each activity. It is useful when we want to break the loop to continue to the next activity in the For each activity.

Example:

Perform the following steps:

1. Add a **Sequence** activity to the **Designer** panel.

2. Next, add a **For each activity** inside the **Sequence** (to use the **Break** activity, we need the **For each activity**)

3. Create two variables; an integer variable named item, and an array integer variable named **x**. Then, set them to the text field.

4. Now, assign a default value to the integer variable x.

5. Add a **Break** activity inside the body of the loop.

6. Under the **For Each** activity, add a **Write line** activity.

7. In the **Write line** activity, type ***item.ToString*** in the text field.

8. When we click the **Run** button, it will display one element. This is due to the Break activity, which has stopped execution after the first iteration.

The While activity:

- The While activity is used in automation to execute a statement or process based on a certain condition.
- If found true, the loop is executed; that is, the process is executed repeatedly. The project only exits from the loop when the condition does not hold true.
- This activity is useful while iterating through an array of elements.

Example:

In the following example, we will see how an integer variable will increase from 5 to 50 in increments of 5.

Perform the following steps:

1. On a **Blank** project, add a **Sequence** activity.

2. Now, create an integer type variable Y. Set its default value to 5.

3. Next, add a **While** activity to the **Sequence**.

4. In the condition field, set x<50.

5. Add an **Assign** activity to the body section of the **While** loop.

6. Now, go to the **Properties** panel of the **Assign** activity and type in the text field integer variable for value field integer x+5

8. Drag and drop a **Write line** activity and specify the variable name x and apply ToString method on this variable

The Do while activity:

- The Do while activity is used in automation when it is required to execute a statement based on the fulfillment of a certain condition.
- It differs from the While activity is that it executes a statement, then checks whether the condition is fulfilled. If the condition is not fulfilled, it exits the loop.

Example: Let us take an example to understand how the Do while activity works in automation. Take an integer variable. Starting with this variable, we shall generate all multiples of 2, less than 20.

Perform the following steps:

1. Add a **Sequence** to the Designer panel.
2. Add a **Do while** activity from the **Activities** panel.
3. In the body section of the **Do while** activity, add an **Assign** activity.
4. Now, select the **Assign** activity. Go to the **Properties** panel and create an integer variable y. Set its default value to 2.
5. Set y+2 in the value section of the **Assign** activity to increment the result each time by until the loop is executed.
6. Add a **Write line** activity inside the **Assign** activity.
7. In the text field of the **Write line** activity, type Z.
8. In the condition section, set the condition y<20. The loop will continue until the condition holds true.
9. On clicking the Run button, the output for each iteration will be displayed.

The For each activity:

- The For each activity works by iterating each element from the collection of items or list of elements, one at a time.
- It will execute all the actions that are available inside the body. Thus, it iterates through the data and processes each piece of information separately.

Example:

In the following example, we shall use the For each activity to go through a collection of even numbers and display each element one at a time.

Perform the following steps:

1. Start with a **Blank** project in UiPath.

2. Add a **Sequence** activity to the Designer panel.
3. Next, add a **For each** activity within the **Sequence** and create an integer type array variable, Y.
4. In the default value of the variable, put in ({2,4,6,8,10,12,14,16,18,20}).
5. Add a **Write line** activity to the Designer Panel
6. In the Text field of the **Write line** activity, type **item.ToString** to display the output
7. Now, run the program. You will see that each number of the array is displayed one by one because of the use of the **For each** activity

The If activity:

- The If activity consists of a statement with two conditions: true or false.
- If the statement is true, then the first condition is executed; if not, the second condition is executed. This is useful when we have to take decisions on the basis of statements.

Example:

Perform the following steps:

1. Add a **Flowchart** from the **Activities** panel.
2. Add two **Input dialog** activities. Create two integer variables, **x** and **y**.
3. In the Properties panel, change the label name and title name of both the **Input dialog** activities.
4. Now, specify these name of these two variables in the **Result** property of both the **Input dialog** activities.
5. Now add the **If activity** to the Designer panel
6. In the condition part, x+y<6, check whether it is ***true*** or ***false***. Add two **Write line activities** and type “True” in one and “False” in the other.
7. Click the **Run** button to check the output. If the condition holds true then it will show the true value; otherwise, it will show the false value. (in our case, we put in the values of x and y as 9 and 4, respectively, thus getting a sum of 13, which is not less than 6; hence, the output shows it as false value)

The Switch activity:

- The Switch activity can be used to make a choice. When we have various options available and want to execute one option, we frequently use the Switch activity.
- By default, the Switch activity takes an integer argument.
- If we want to take a desired argument, then we can change it from the Properties panel, from the **TypeArgument** list.
- The Switch activity is very useful in the categorization of data according to one's own choice.

Example

Let's see an example where we have to check whether a given number is odd or even.

Perform the following steps:

1. Add a **Sequence** activity.
2. Add an **Input dialog** activity inside the **Sequence**.
3. Now, create an integer type variable k.
4. Specify the newly created variable's name in the **Result** property inside the **Properties** panel.
5. Add the **Switch** activity under the **Input dialog** activity.
6. In the **Expression** field, set k mod 2 to check whether the number is divisible by 2 or not.
7. Add a **Write line** activity to the **Default** section and type the k.ToString +”is an even number” in the text field.
8. Now, create **Case 1**, add the one other **Write line** activity to it, and type k.ToString +”is an odd number” in the text field

Step-by-step example using Sequence and Control flow:

Consider an array of names. Say we have to find out how many of them start with the letter a. We will then create an automation where the number of names starting with a is counted and the result is displayed.

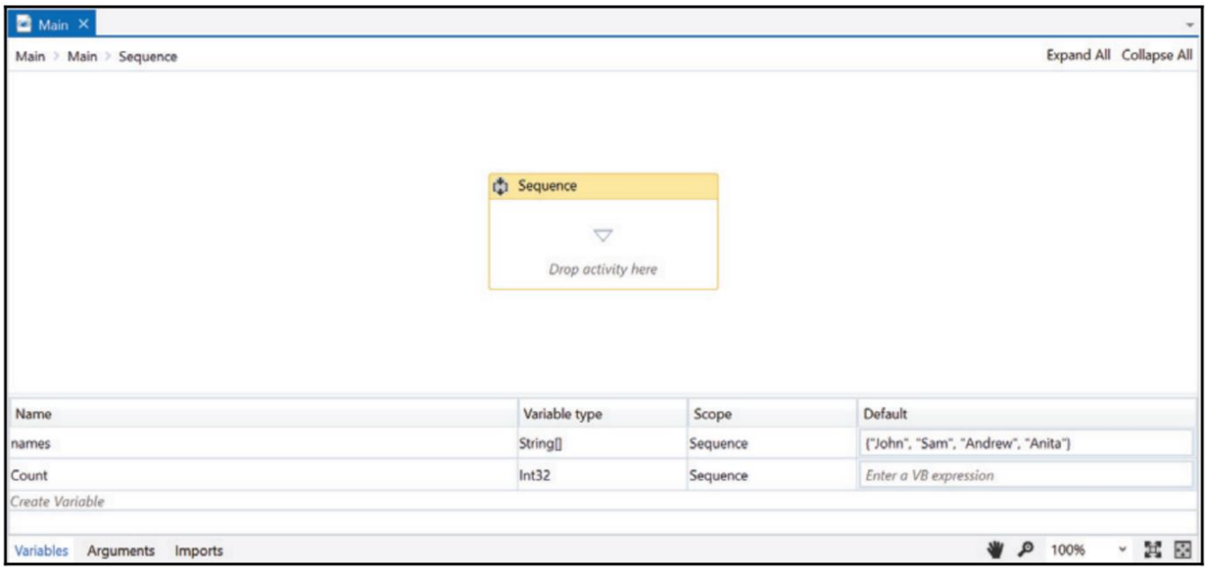
Perform the following steps:

1. Drag and drop a **Flowchart** activity from the **Activities** panel.
2. Drag and drop a **Sequence** activity inside the **Flowchart**. Connect the **Sequence** to the **Start** node by right-clicking on the **Sequence** activity and selecting the **Set as Start node** option.
3. Double click on the **Sequence** activity. Create a variable. Give it a name (in our case, we will create an array of type string and name the variable as names). Set the variable type to

Array of [T]. When asked for the type of array, select **String**. Also, initialize the array in the **Default** section of the variable by giving it a default values.

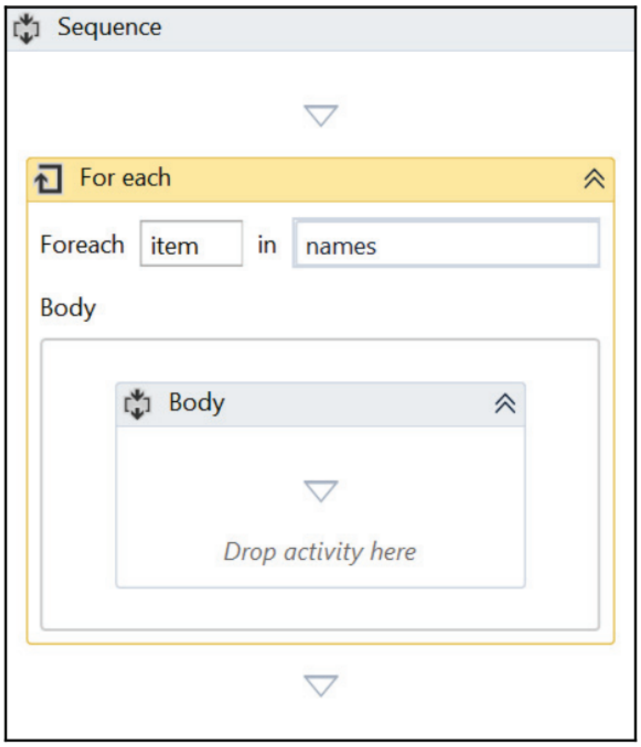
For example, {“John”,”Sam”,”Andrew”,”Anita”}.

4. Create a variable of type integer **Count** for storing the result. Set the variable type to **Int32**



Name	Variable type	Scope	Default
names	String[]	Sequence	{"John", "Sam", "Andrew", "Anita"}
Count	Int32	Sequence	Enter a VB expression

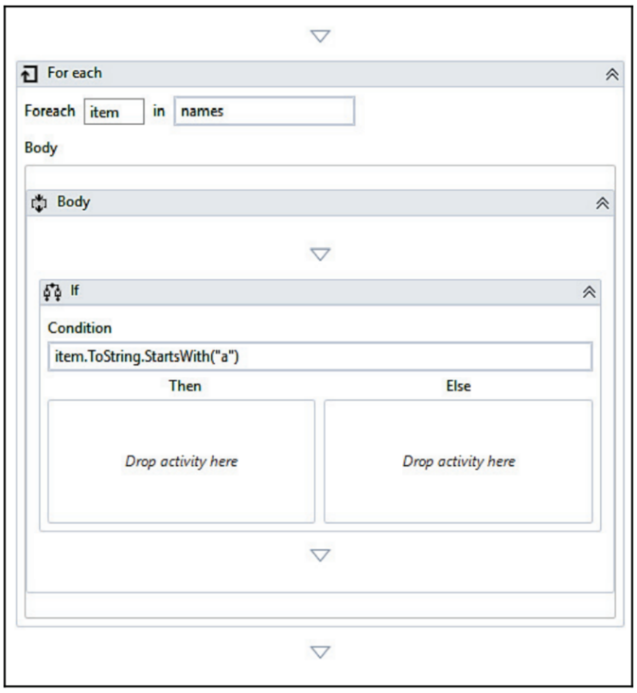
5. Drag and drop a **For each** activity inside the **Sequence**. Also, specify the array name in the expression box of the **For each** activity. The **For each** activity is used to iterate over the array. It will pick up one name from the array each time until it reaches the end:



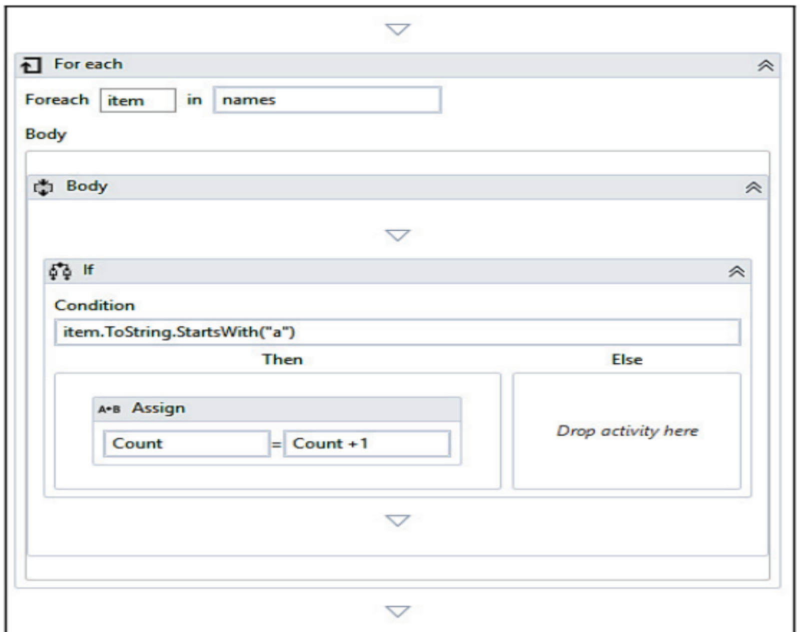
Foreach item in names

6. Drag and drop the **If** activity from the **Activities** panel and place it inside the **For each** activity at the location where ***Drop activity here*** is mentioned. Specify the condition in the expression box of the **If** activity. The **If** activity is used to check for a particular condition/expression. If that expression is satisfied, the **Then** block will be executed. Otherwise, the **Else** block will be executed.

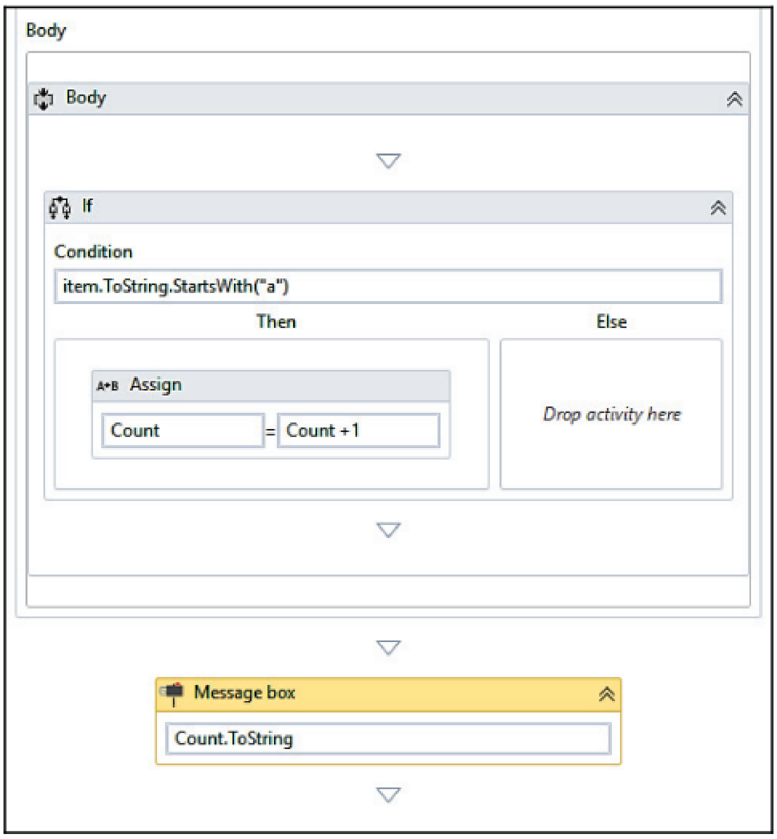
   We have specified the expression as **item.ToString,StartsWith('a').** This expression specifies the name present in the item variable starts with the letter 'a' . The **For each** activity iterates over the array, picks up one name at a time, and stores it as a variable, **item**:



7. Now, we are going to use the **Count** variable and increment it each time a name from an array starts with the letter **a**. For this, we have to use the **A+B Assign** activity. Drag and drop the **A+B Assign** activity inside the **If** activity. Set the **To** property to **Count** (variable name) and the **Value** property to Count+1 (to increment its value) of the **A+B Assign** activity:



8. Just drag and drop a **Message box** activity inside the **Sequence** activity. Specify the count variable in the expression box of the **Message box** activity. But remember, the variable that we have created is of type **Int32**, so, it cannot be used with the **Message box** activity without converting it to a string. To convert it to a string, we have the **'.toString'** method available in UiPath Studio. Just apply it to the variable and select **'.toString'**:

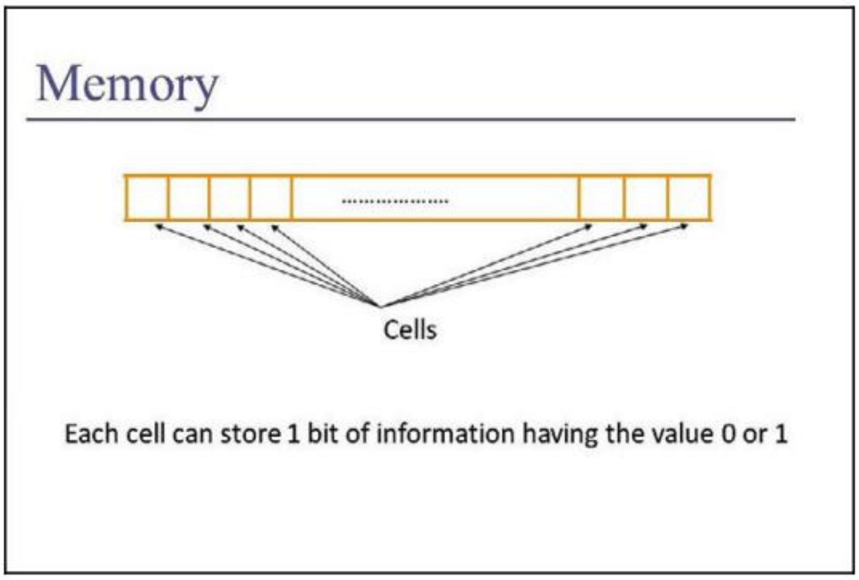


```
item.ToString.StartsWith("a")
Count = Count + 1
Count.ToString
```

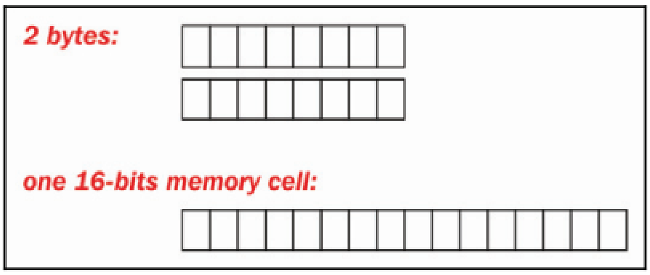
Hit the **Run** button or press **F5** and see the result.

Data Manipulation

Variables and scope:


Memory
....
Cells
Each cell can store 1 bit of information having the value 0 or 1

Memory consists of millions of memory Cells and each memory cell stores data in the form of 0s and 1s (binary digits). Each cell has a unique address, and by using this address, the cell can be accessed:



When data is stored in memory, its content gets split into further smaller forms (binary digits). As shown in the preceding diagram, 2 bytes of data consists of several memory cells.

A variable is the name that is given to a particular chunk of memory cells or simply a block of memory and is used to hold data.

You can declare any desired name and create a variable to store the data. It is recommended, however, that we use meaningful variable names.

For example, if we wish to create a variable to store the name of a person, then we should declare

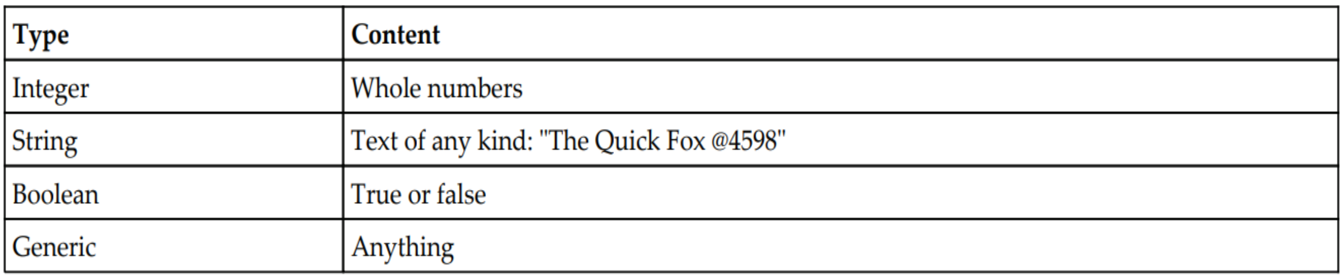
Name: Andy

It is a good practice to create meaningful variable names. This becomes very useful in debugging the program.

A variable is used to store data. Data is present around us in different types - it can be an mp3 file, text file, string, numbers, and so on. That is why variables are associated with their respective data types.

A particular type of variable can hold only that type of data. If there is a mismatch between the data and the variable type, then an error occurs.

The following table shows the type a of variable available with UiPath:



Type	**Content**
Integer	Whole numbers
String	Text of any kind: "The Quick Fox @4598"
Boolean	True or false
Generic	Anything

In UiPath, we can declare a variable in the Variables section. Just give it a meaningful name and select the appropriate type from the drop-down list.

We can also specify the scope of a variable. The Scope is the region under which the data has its effect or availability. We can choose the Scope of the variable according to your requirements; try to limit it as far as possible.

Collections:

There are different types of variables. Variables can be classified into three categories:

- **Scalar**: These are variables that can only hold a single data point of a particular data type, for example; Character, Integer, Double, and so on.

- **Collections**: These are variables that can hold one or more data point of a particular data type. For example; array, list, dictionary, and so on.
- **Tables**: These are a tabular form of the data structure which consists of rows and columns.

In a collection, we can store one or more data points, but all the data must be the same. Consider an example. An array is a collection in which we can store different values of a particular data type. It is a fixed data type, meaning if we store five values inside the array, we cannot add or remove any value/values in that array.

Let us see how we can use an array with an example. In this example, we are going to take an array of integers, initialize it, and then iterate through all the elements of the array:

1. Drag and drop a **Flowchart** activity onto the main Designer panel, and drag and drop a **Sequence** activity inside the **Flowchart**. Set the sequence as **Start** node.

2. Create a variable in the **Variables** panel and give it a meaningful name (in this example, we have created a variable named **arr**, which is an array of integers). Choose the data type as an array of integers.

3. Initialize the array as {1,2,3,4,5} in the **Default** section. You can initialize it with the int32 data type.

4. Drag and drop a **For each** activity from the **Activities** panel inside the **Sequence**, and drag and drop a **Message box** activity inside the **For each** activity.

5. Specify the array name in the expression text box of the **For each** activity.

6. Specify the **item** variable that is auto-generated by the **For each** activity, inside the **Message box** activity. We have to convert the **item** variable into the **String** type because the **Message box** activity is expecting the string data type in the text box. Just press the dot (.) along with the **item** variable and choose the **ToString** method.

Hit the Run button to see the result. All the values will pop up at once.

Arguments - Purpose and use:

An Argument is simply a variable that can store a value. We can create an argument in the Argument section of the main Designer panel.

An argument has a larger scope than a variable and is used to pass values between different workflows. We might be wondering why we need this. Suppose we have a big project to build; we break down the project into different workflows because smaller workflows can be easily tested separately. It is very easy to build smaller workflows and combine them, thus turning them into the real solution of the project.

These Arguments are used for interacting with different workflows by exchanging data between them. That is why the direction property is associated with Arguments. We can choose the direction on the basis of our requirement-either giving the value to some workflow or receiving the value from another workflow.

We can easily create arguments in the **Arguments** panel. We can also specify the direction:

- **In**: When we have to receive the value from another workflow.
- **Out**: This is the current value if we have to send the value to a workflow.
- **In/Out**: This specifies both; it can take or receive the value.
- **Property**: This specifies that it is not being used currently

Data table usage with examples:

A data table is a tabular form of data structure. It contains rows and each row has columns, for example:

Student name	**Roll number**	**Class**
Andrew Jose	1	3
Jorge Martinez	2	3
Stephen Cripps	3	2

A data table is used for various purposes. For example, you have to build a table dynamically. You can use a data table as your preferred choice. A data table is also extensively used to store tabular data structures. In data scraping, data tables are widely used. Data scraping is a method in which we can dynamically create tabular data records of search items on the web.

Building a data table:

First, create an empty project. Give it a proper name:

1. Drag and drop a **Flowchart** activity on the **Designer** panel. Also, drag and drop a **Sequence** activity and set it as the **Start** node.

2. Double click on the **Sequence** and drag and drop the **Build Data Table** activity inside the **Sequence** activity.

3. Click on the **Data Table** button. A pop-up window will appear on the screen. Remove both the columns (auto generated by the **Build Data Table** activity) by clicking on the Remove Column icon

4. Add three columns by simply clicking on the + symbol. Specify the column names and select the appropriate data types from the drop-down list. Click on the **OK** button. We will add column ***Name*** of **String** Data Type, ***Roll_No*** of **Int32** type and finally ***Class*** of **String** type:

Now enter some values just to insert the data into the rows. Click on the **OK** button and our data table is ready. We have to iterate over the data table's rows to make sure everything works correctly.

5. In order to store the Data Table created by **Build Data Table** activity, we have to create a data table variable ***data*** of **DataTable** type and in order to store the result of the data table that we have dynamically built. Also, specify assign the **Output** property of the **Build Data Table** activity with this variable. Specify the data table variable's name there.

6. After our data table is ready, we will iterate the data table's rows to make sure everything works correctly. Drag and drop the **For each row** activity from the **Activities** panel inside the **Sequence** activity. Specify the data table variable's name (***data***) in the expression text box of the **For each row** activity

7. Drag and drop a Message box activity inside the For each row activity. In the Message box activity, Inside the message box we have to write following string: ***row("Name").ToString+"-"+ row("Roll_No").ToString+"-"+ row("Class").ToString.***

row *is* the variable which holding data for the data row in each iteration

This ***row*** variable contains all the columns of a particular row. Hence, we have to specify which column value we want to retrieve by specifying the column name. Instead of the column name, we can also specify the column index

Hit the ***Run*** button to see the result.

Building a data table using data scraping (dynamically):

Using data scraping, we can build the data table at runtime. Let us consider an example of extracting data from Amazon's website. Perform the following steps:

1. Drag and drop the **Flowchart** activity from the **Activities** panel, and drag and drop the **Sequence** activity inside the **Flowchart** activity.

2. Double-click on the **Sequence** activity.

3. Drag and drop the **Open Browser** activity inside the **Sequence** activity. Specify the URL in the text box

(Ex: https://www.amazon.in/s?k=books+for+kids&crid=2C1Z8GTXI7ZP9&sprefix=books+for+kids%2Caps%2C223&ref=nb_sb_noss_1)

4. Click on the **Data Scraping** icon on the top left corner of UiPath Studio. A window will pop up. Click on the **Next** button.

5. Now, there will be a pointer pointing to the UI elements of the web page. Click on the name of the book. It will ask you to point to a second similar element on the web page

6. Point to a second similar element on that web page. Specify the name that you want to give for that extracted data column. (It will become the column name of the extracted data). Click on the **Next** button.

7. A list of names will appear in a separate window. If you want to extract more information, then click on the **Extract correlated data** button and repeat the same process once again (just

as we extracted the name of the book from Amazon's website). Otherwise, click on the **Finish** Button

8. It will ask you to locate the next page's button/link. If you want to extract more information about the product and it spans across multiple pages, then click on the **Yes** button and point to the next page's button/link. Then, click on it. If you want to extract only the current page's data, click on the **No** button, (you can also specify the number of rows that you want to extract data from: By default it is 100)

9. Data scraping generates a data table. (In this case, **ExtractedDataTable** is generated.) Change the scope of **ExtractedDataTable** to the **Flowchart** so that it is accessible within the **Flowchart** activity

10. Drag and drop the **Output data table** activity on the **Flowchart**. Set the **Output** property of the **Output data table** activity as: **ExtractedDataTable**

11. Connect the **Output data table** activity to the **Data Scraping** activity. Drag and drop the **Message box** activity on the Designer window. Also create a string variable to receive the text from the **Output data table** activity (in our case, we have created a ***result*** variable). Specify the text property of the **Output data table** activity as the ***result*** variable to receive the text from the **Output data table**

12. Connect the **Message box** activity to the **Output data table** activity. Double-click on the **Message box** and specify the text property as the ***result*** variable (the variable that you created to receive the text from the **Output data table** activity).

13. Hit the **Run** button and see the result.

Clipboard management:

Clipboard management involves managing the activities of the clipboard, for example, getting text from the clipboard, copying selected text from the clipboard, and so on.

Let us see an example of getting text from the clipboard. In this example, we will use Notepad.

We will open Notepad, write some data into it, and then copy the data to the clipboard. We will then extract the data from the clipboard:

1. Drag and drop a **Flowchart** activity from the **Activities** panel.

2. Click on the **Recording** icon on the top of UiPath Studio. A drop-down menu will appear with the options, **Basic, Desktop, Web,** and **Citrix**, indicating the different types of recording. Select **Desktop** and click on **Record**.

3. Click on **Notepad** to open it. A Notepad window will pop up:

4. Click on the text area of **Notepad**. Type into the dialog box and check the empty field. (Checking the empty field will erase all existing data in Notepad before writing any new data.) Press ***Enter***. Data will be written on the **Notepad** text area

5. Click on the **Edit** button. A pop-up window will appear asking you whether you want to use an anchor. (An anchor is a relative element of the current {***focussed***} element.) As you can see clearly, the anchor element of the **Edit** button can be the **File** or **Format** button. In this case, we have chosen the **Format** button:

6. Then, it will automatically start recognizing the Edit button. Choose the **Select all** option from the drop-down list

7. Once again, click on the **Edit** button. It will again ask you to indicate the anchor element. Indicate the anchor button and the **Edit** button will be highlighted, giving you a drop-down box. Select the **Copy** option

This copied text is now stored in the clipboard. We can use the **Get from clipboard**, and **Copy selected text** activities to copy the text that is stored in the clipboard. We will use the **Copy selected text** activity.

8. Double-click on the **Recording sequence** that is generated by the recording. Scroll down and drag and drop the **Copy selected text** and **Message box** activities inside the **Recording sequence**

9. Create a variable of type **String** to store the output value of **Copy selected text**. This variable will receive the required text from the clipboard with the **Copy selected text** activity. Now,

specify the newly created variable in the **Output** property of the **Copy selected text** activity. This will be the required selected text that we have copied into the clipboard.

10. Specify the string variable in the text property of the **Message box** activity.

11. Hit the **Run** button to see the result.

Methods that are frequently used with an Excel file:

The following are the methods that are frequently used with an Excel file:

- Read cell
- Write cell
- Read range
- Write range
- Append range

Read cell :

This is used to read the value of a cell from an Excel file.

Consider a Sample Excel file that contains values as shown below:

Suppose we have to read the value of the B3 cell:

1. Drag and drop a **Flowchart** activity on the main Designer panel. Also, drag and drop an **Excel application scope** inside the **Flowchart**. Connect it to the **Start** node. Double click on Excel application scope.
2. Drag and drop the **Read Cell** activity inside the **Excel application scope** activity. Specify the range value in the cell text box of the **Read Cell** activity. Create a variable of type string to hold the result produced by the **Read Cell** activity. In our case, we have created a Result variable. Specify the Output property of the Read Cell activity by providing the variable's name that we have created
3. Drag and drop a **Message box** activity inside the **Excel application scope** activity and specify the string variable's name in the expression box of the Message box activity. That's it. Press F5 to see the result.

Write cell:

This activity is used to write a value in a cell of an Excel file:

1. Drag and drop a **Flowchart** activity on the main Designer panel. Also, drag and drop an **Excel application scope** inside the **Flowchart** activity. Connect it to the **Start** node.

2. Drag and drop a **Write Cell** activity inside the **Excel application scope**. Specify the cell value in which we want to write in the **Range** property of the **Write Cell** activity. Also, specify the value of the **Value** property:

Press F5 and see the result. Open the Excel file to see the changes.

Read range:

This is used to read the value up to the specified range. If the range parameter is not specified, it will read the entire Excel file:

1. Drag and drop a **Flowchart** activity on the main Designer panel. Also, drag and drop an **Excel application scope** inside the **Flowchart** activity. Connect it to the Start node.

2. Drag and drop a **Read Range** activity inside the **Excel application scope** activity. The **Read Range** activity produces a data table. We have to receive this data table in order to consume it. We need to create a data table variable and specify it in the **Output** property of the **Read Range** activity.

3. Drag and drop an **Output Data Table** activity inside the **Excel application scope** activity. Now, we have to specify two properties of the **Output Data Table** activity: **Data Table** property and text property. The **Data Table** property of the **Output Data Table** activity is used to convert the data table into a string format. The text property is used to supply its value in a string format. We have to receive this value in order to consume it. For this, let us create a variable of type string. Give it a meaningful name

4. Drag and drop a **Message box** activity inside the **Excel application scope** activity. Also, specify the string variable's name that we created earlier inside the **Message box** activity:

Press F5 to see the result. A window will pop up displaying your Excel file data.

Write range:

This is used to write a collection of rows into the Excel sheet.

It writes to the Excel file in the form of a data table. Hence, we have to supply a data table:

1. Drag and drop a **Build data table** activity from the **Activities** panel. Double-click on this activity. A window will pop up. You will notice that two columns have been generated automatically. Delete these two columns. Add your column by clicking on the + icon and specify the column name. You can also select your preferred data type. You are free to add any number of columns:
2. In this project, we are adding two columns. The procedure for adding the second column is almost the same. You just have to specify a name and its preferred data type. We have added one more column (**Roll**) and set the data type to **Int32** for the data table. We have also initialized this data table by providing some values in its rows. Create a variable of type data table. Give it a meaningful name. Specify this data table name in the **Data table** property of the **Build data table** activity. We have to supply this variable in order to get the data table that we have built:

   Our data table has been built successfully.

3. Drag and drop an **Excel application scope** inside the main Designer panel. You can either specify the Excel sheet path or manually select it. Connect this activity to the **Build Data Table** activity. Inside the **Excel application scope** activity, just drag and drop the **Write Range** activity:
4. Specify the data table variable name that we created earlier and set it as a **Data table** property inside the **Write Range** activity. We can also specify the range. In this case, we have assigned it as an empty string:
5. Hit the Run button or press F5 to see the result.

Append range:

This is used to add more data into an existing Excel file. The data will be appended to the end.

1. Drag and drop the **Flowchart** activity on the main Designer window. Also, drag and drop the **Excel application scope** inside the **Flowchart** activity. Connect it to the Start node.

The **Append Range** activity requires a data table. In this program, we are going to use another sample Excel file, which has some raw data. Then, we will read this Excel file and append the data to another Excel file.

First, we have to read its contents:

2. Drag and drop the **Read Range** activity inside the **Excel application scope** activity. The **Read Range** activity produces a data table. We have to receive this data table in order to consume it. Create a data table variable and specify it in the **Output** property of the **Read Range** activity:

3. Drag and drop the **Append Range** activity inside the **Excel application scope** activity. Specify the Excel file path in the **Append Range** activity (in which we want to append the data). Also, specify the data table (which is generated by the **Read Range** activity):

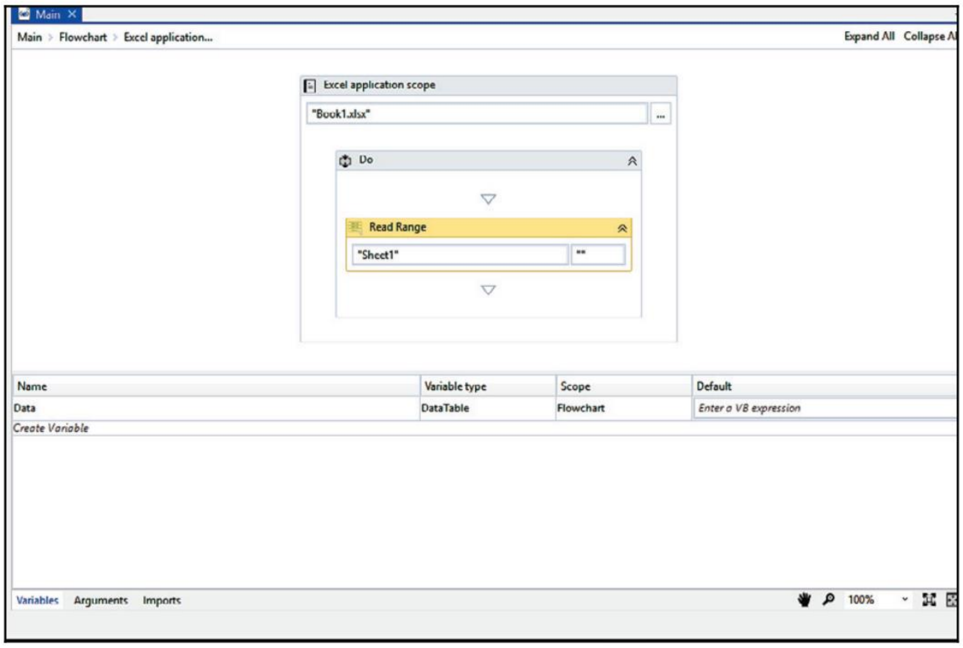
Press F5 to see the result:

Reading an Excel file and creating a data table by using data from the Excel file:

We have an existing Excel file and we are going to use it in our project:

1. Drag and drop the **Flowchart** activity on the main Designer window. Also, drag and drop the **Excel application scope** inside the **Flowchart**.

2. Double-click on **the Excel application scope**. You have to specify the path of your workbook/Excel file. Drag and drop the **Read Range** activity from the **Activities** panel inside the **Excel application scope**.

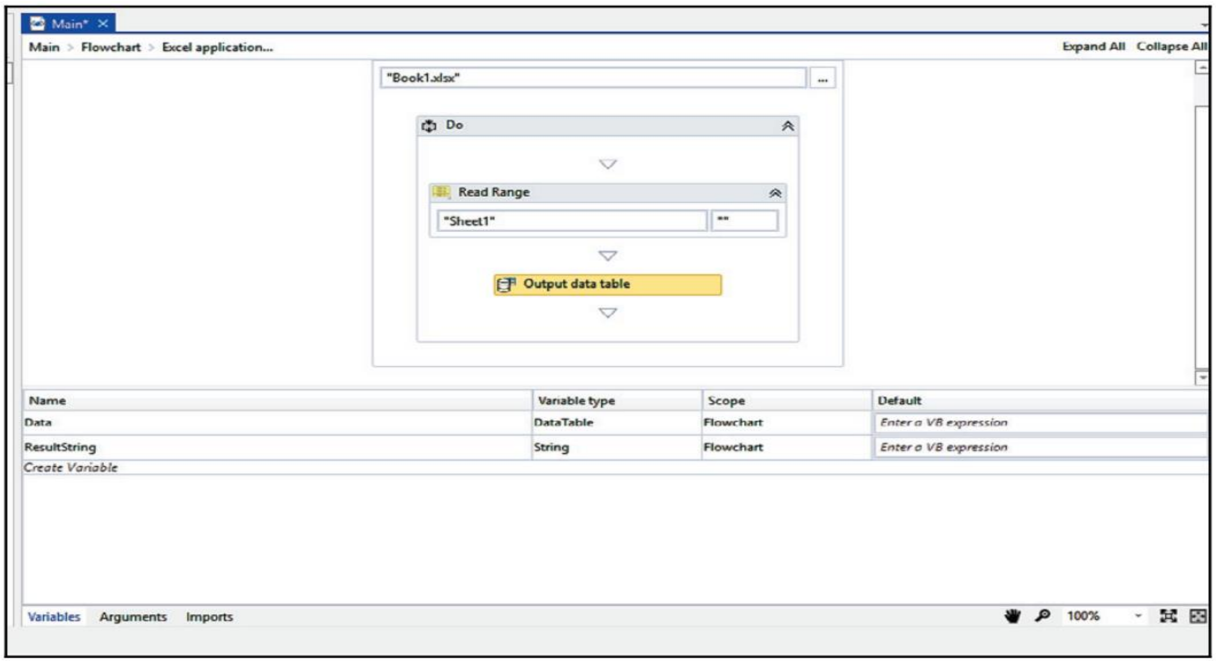


Name	Variable type	Scope	Default
Data	DataTable	Flowchart	Enter a VB expression
Create Variable			

The **Read Range** activity will read the entire Excel sheet. We also have the option of specifying our range. Create a variable of type data table and specify it in the **Output** property of the **Read Range** activity. This variable will receive the data table produced by the **Read Range** activity:

3. Drag and drop the **Output Data Table** activity inside the **Excel application scope** activity. Now, we have to specify two properties of the **Output Data Table** activity: the **Data Table** property and the text property. The **Data Table** property of the **Output Data Table** activity is used to convert the **Data Table** into string format.

   The text property is used to supply its value in a string format. We have to receive this value in order to consume it. For this, let us create a variable of type string. Give it a meaningful name:



Name	Variable type	Scope	Default
Data	DataTable	Flowchart	Enter a VB expression
ResultString	String	Flowchart	Enter a VB expression
Create Variable			

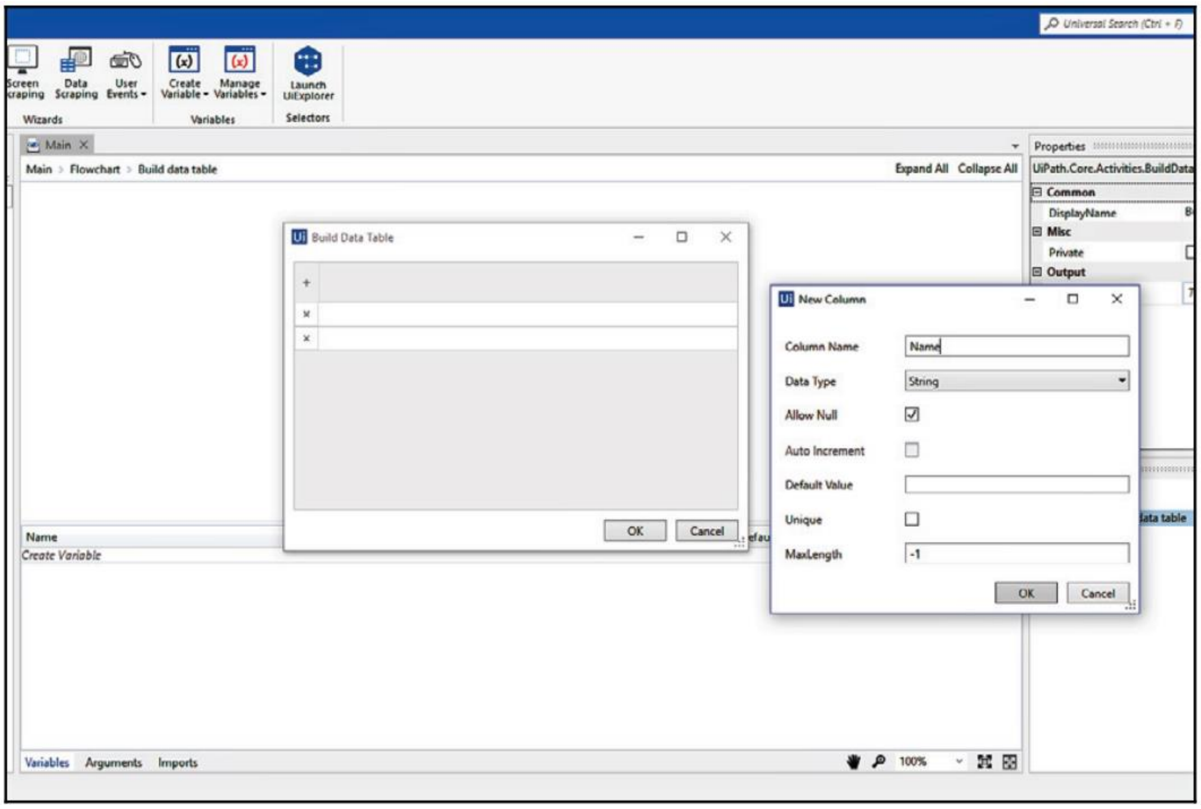
4. Drag and drop a **Message box** activity inside the **Excel application scope** activity. Also, specify the string variable's name that we created earlier inside the **Message box** activity.

Press **F5** to see the result. A window displaying the Excel file data will pop up.

Creating a data table and then writing all its data to an Excel file:

In this project, we will build a data table dynamically and then write all its data to an Excel file:

1. Drag and drop a **Build data table** activity from the **Activities** panel. Double-click on this activity. A window will pop up. Two columns have been generated automatically; delete these two columns. Add your column by clicking on the + icon and specify the column name. You can also select your preferred data type. You are free to add any number of columns:



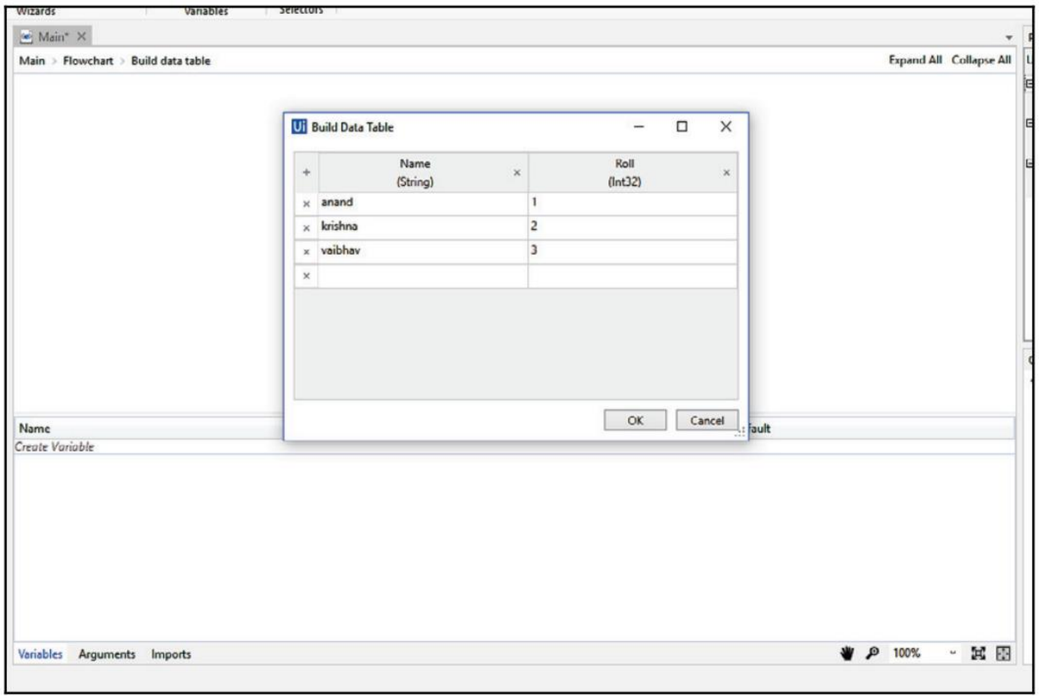
Column Name: Name
Data Type: String
Allow Null: ☑
Auto Increment: ☐
Default Value:
Unique: ☐
MaxLength: -1

OK Cancel

2. In this project, we are adding two columns. The procedure for adding the second column is almost the same. You just have to specify a name and its preferred data type.

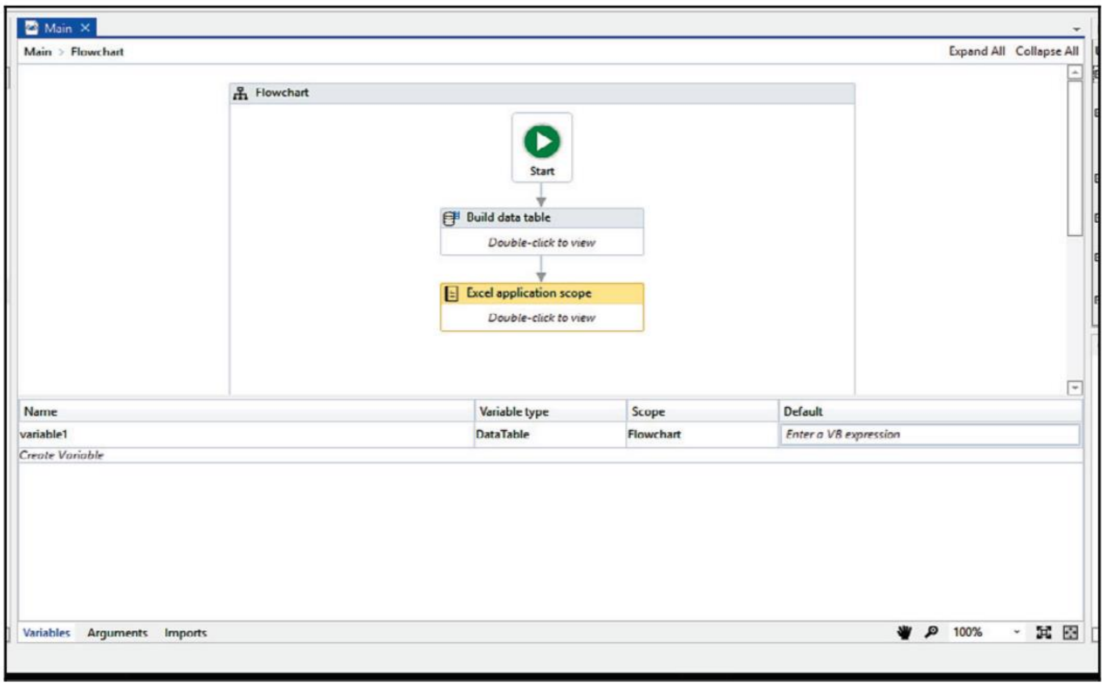
We have added one more column (Roll) and set the data type to **Int32** in the data table. We have also initialized this data table by giving some values to its rows.

Create a variable of type **Data Table**. Give it a meaningful name. Specify this data table's name in the **Data Table** property of the **Build data table** activity. We have to supply this variable in order to get the data table that we have built:



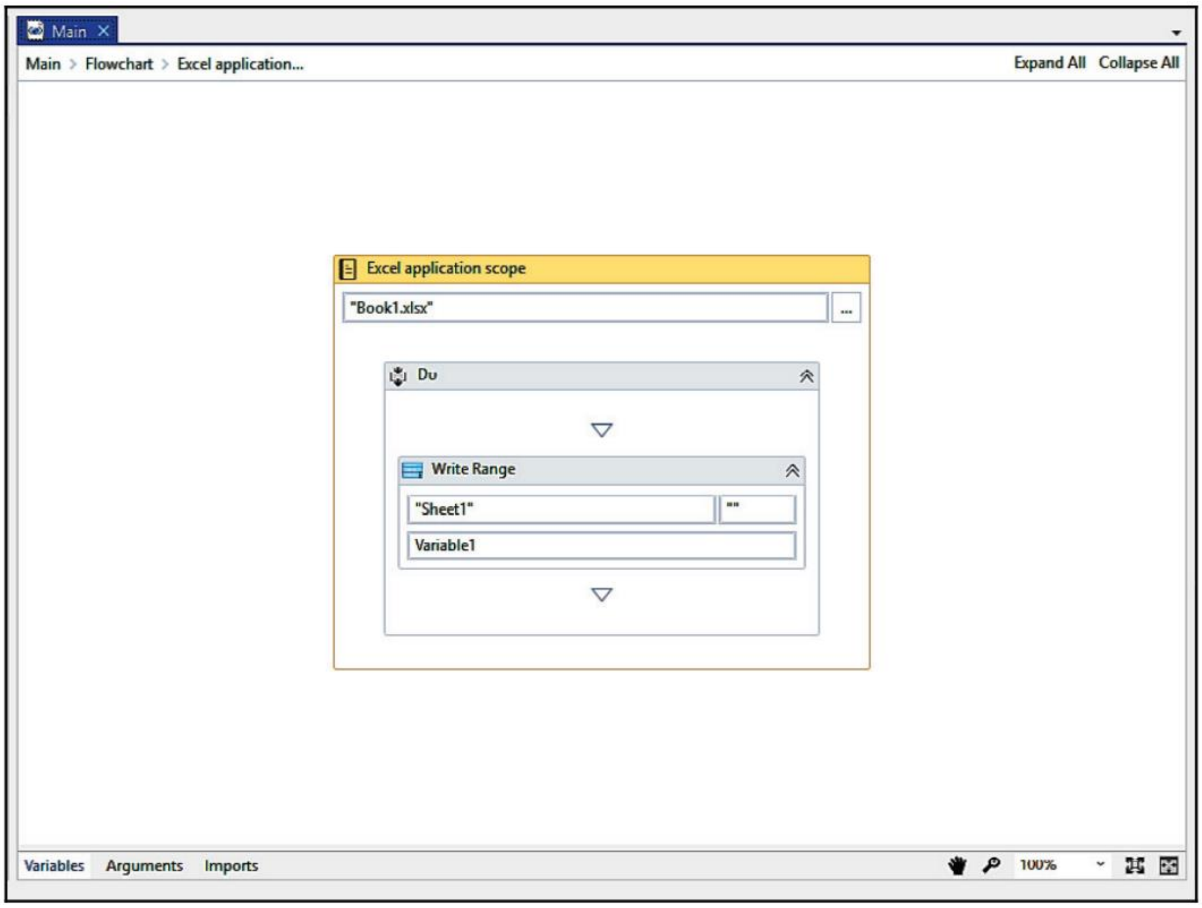
Name (String)	Roll (Int32)
anand	1
krishna	2
vaibhav	3

3. Drag and drop the **Excel application scope** inside the main Designer window. Specify the Excel sheet's path or manually select it. Connect this activity to the **Build Data table** activity:



Name	Variable type	Scope	Default
variable1	DataTable	Flowchart	Enter a VB expression

4. Inside the **Excel application scope** activity, drag and drop the **Write Range** activity. Specify the data table variable name that we created earlier and set it as a **Data table** property inside the **Write Range** activity. We can also specify the range. In this case, we have assigned it as an empty string:



5. That's it. Hit the **Run** button or press F5 to see the result.