

# RPA Design and Development V3.0

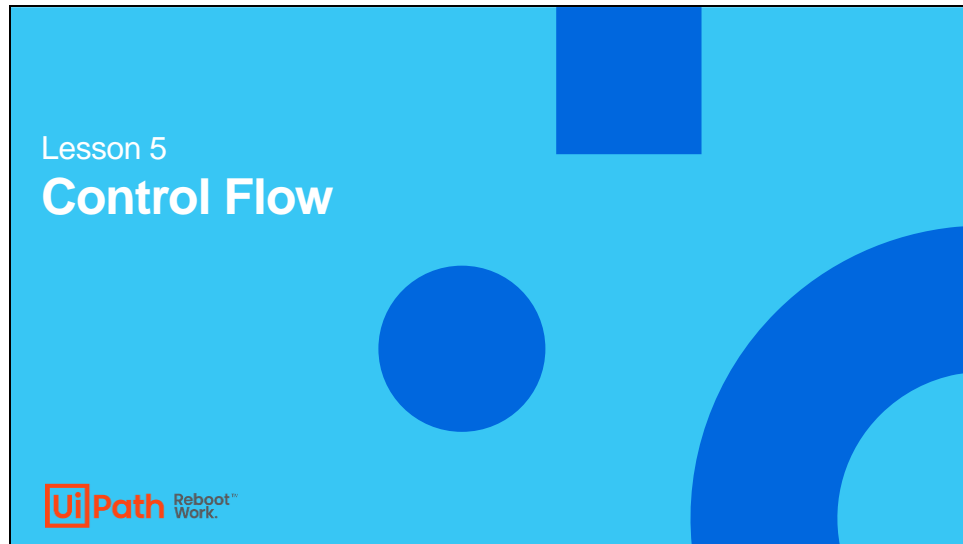
STUDENT MANUAL – Lesson 5

Slide 1



Welcome to 'RPA Design and Development Course'.

Slide 2



The fifth lesson of this course is Control Flow.

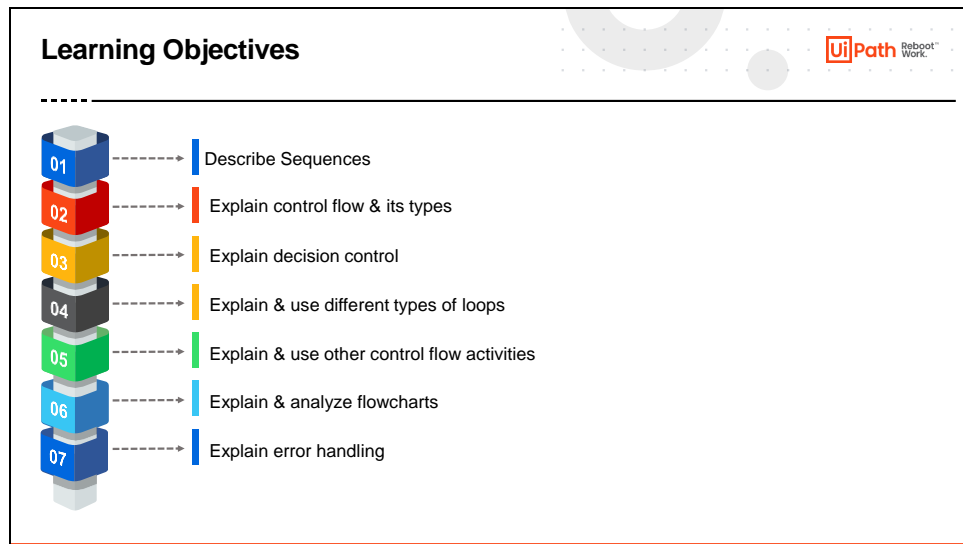
## Slide 3

Agenda	
01	Sequences
02	Control Flow and Its Types
03	Decision Control
04	Loops
05	Other Control Flow Activities
06	Flowcharts
07	Error Handling

The agenda of this lesson is:

- Sequences
- Control Flow and Its Types
- Decision Control
- Loops
- Other Control Flow Activities
- Flowcharts
- Error Handling

## Slide 4



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

- Describe Sequences
- Explain control flow & its types
- Explain decision control
- Explain & use different types of loops
- Explain & use other control flow activities
- Explain & analyze flowcharts
- Explain error handling

Slide 5



This section gives an overview of Sequences.

## Slide 6

## Introduction to Sequences

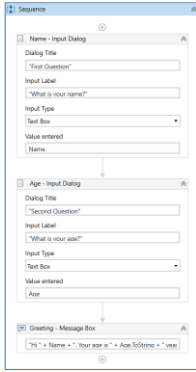
-----

A Sequence is a container in which activities are placed one after another and executed linearly.

**A Sequence:**

- Enables the user to create linear processes
- Enables seamless movement from one activity to another
- Can be reused

Example: A Sequence that asks for the Name and Age of a user and displays a greeting message.



In UiPath Studio, a Sequence is a container in which multiple activities are placed one after another and executed linearly.

Within the Sequence, an activity is followed by the next activity in a linear fashion. The Sequence can contain any number of activities, but no activity can be skipped during the execution. The program, when run, must execute each activity linearly with no possibility of skipping an activity or branching off to another activity.

One of the key features of Sequences is that they can be reused time and again. Also, it enables the seamless movement from one activity to another.

In the example on the slide, the screenshot demonstrates a Sequence that asks the user for the following details:

- Name
- Age


On execution, the workflow will ask for input in the above-mentioned order. A Sequence cannot ask for Age before asking for a Name. It will first ask for Name and then Age and display the results accordingly.


To know more, visit: <https://docs.uipath.com/studio/v2020.10/docs/sequences> for additional detail on this example.

Slide 7

## Control Flow and its Types

- Control Flow
- Types of Control Flow

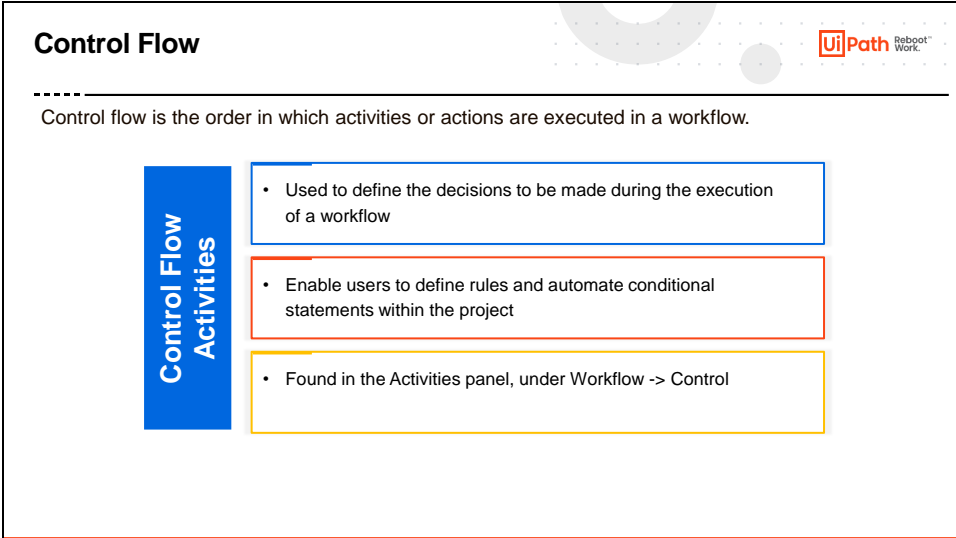




This section explains control flow and gives an overview of its types.



## Slide 8



**Control Flow**

Control flow is the order in which activities or actions are executed in a workflow.

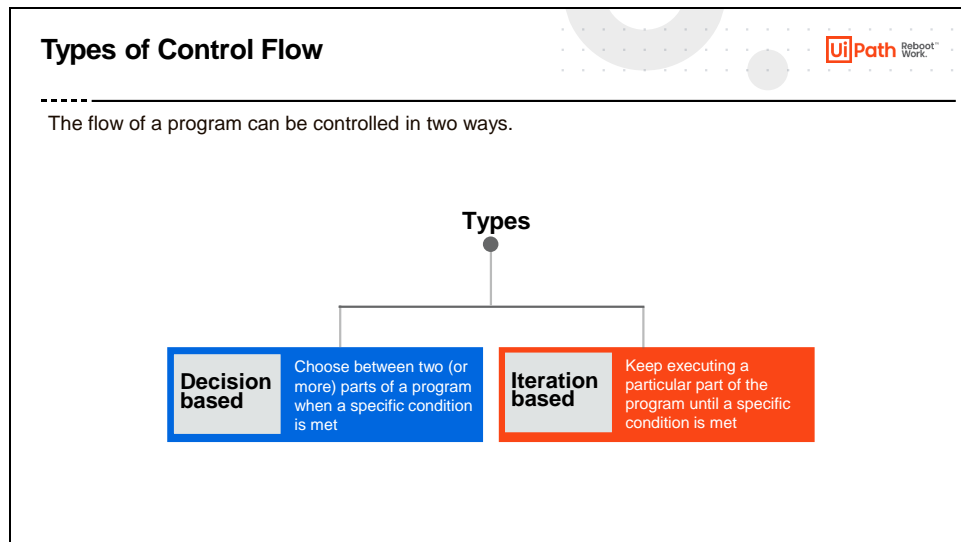
**Control Flow Activities**

- Used to define the decisions to be made during the execution of a workflow
- Enable users to define rules and automate conditional statements within the project
- Found in the Activities panel, under Workflow -> Control

The diagram features a blue vertical bar on the left with the text 'Control Flow Activities'. To its right are three stacked boxes: a blue-outlined box for the first bullet point, a red-outlined box for the second, and a yellow-outlined box for the third. The UiPath logo is in the top right corner of the slide frame.

Control flow is the order in which activities or actions are executed or performed in a workflow. Control flow is enacted through control flow activities which are used to define the decisions to be made during the execution of a workflow. The execution of control flow activities helps the user to choose a suitable path to follow in the workflow. These activities are found in the Activities panel, under Workflow -> Control.

## Slide 9



The flow of an automation project can be controlled in two ways:

- **Decision-based control flow**: Here, a decision is made on the basis of a specified condition. If the condition is met, the program executes one part and if not, then it executes some other part. This consists of **If** and **Switch** activities.
- **Iteration-based control flow**: Here, a particular part of a program is executed many times until a specific condition is met or holds true. This consists of loops such as **While**, **Do While**, **For Each**.

In addition to these two types, there are other activities in UiPath that help the user to control the flow of execution. This includes activities like delay, parallel, assign, etc.


These activities are discussed in detail in the coming sections.

## Slide 10

### Decision Control

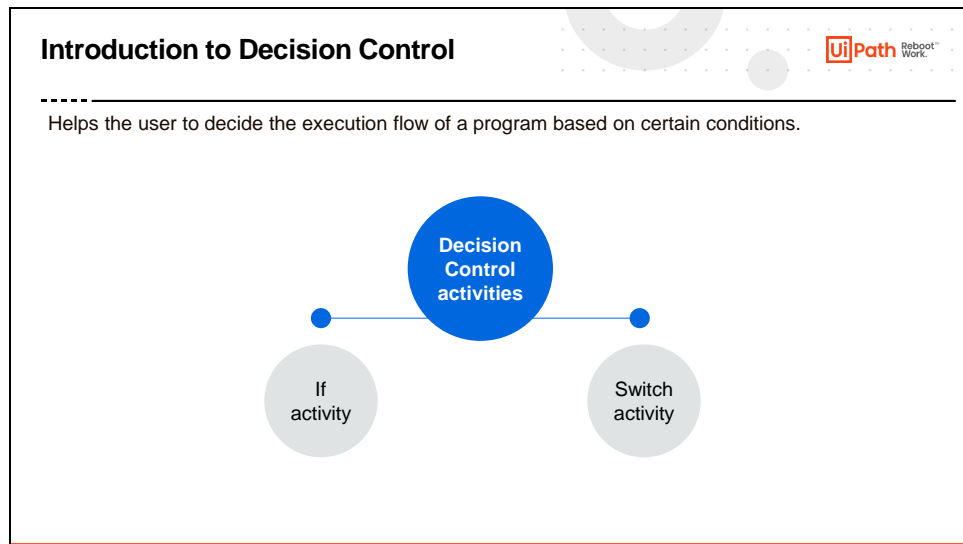
- Introduction to Decision Control
- If Activity
- Switch Activity
- If vs. Switch



**UiPath** Reboot™  
Work.

This section explains decision control in detail.

## Slide 11



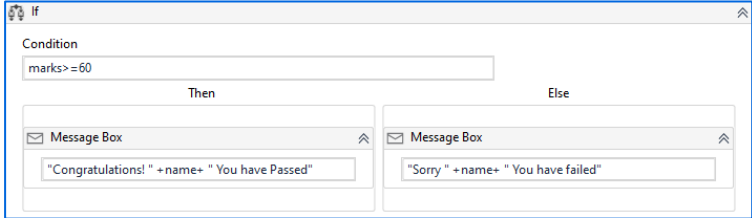
Decision-based control helps the user to decide the execution flow of the program based on certain conditions. These conditions are specified by a set of conditional statements having the Boolean value true or false. Each of these conditions, when met, transfers the control to execute a subroutine. A subroutine is a Sequence of program instructions that performs a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed. Whenever the specified condition is met, one of these subroutines is executed.

There are two types of decision-based activities in UiPath: **If activity** and **Switch activity**.

## Slide 12

### If Activity

Contains a statement with a condition, and two sets of instructions (Then & Else) as outcomes.



The If activity contains a statement with a condition attached, and two sets of instructions as outcomes:

- Then: The set of actions to be executed when the condition is true
- Else: The set of actions to be executed when the condition is false

The If activity is useful to make decisions based on the value of variables.

**Example:**


An automation to input a candidate's name & marks obtained in an exam and decide if he has passed the exam or not. Here, the condition is specified in an If activity which displays the result accordingly:

- if the marks are  $\geq 60$ , then  
**Congratulations! 'name' You have passed**
- Else  
**Sorry, 'name' You have failed**

## Slide 13

## Classroom Exercise

---



Demonstrate the use of **If** statement by building a workflow that tells whether the user has passed the exam or not.

- Take input from a user of name & marks obtained in an exam.
- The passing marks are greater than or equal to 60.
- Display the result in a message box for Pass as "Congratulations! you have passed the exam."
- Display the result in a message box for Fail as "Hello User, you have failed the exam". Replace "User" with the user's name.

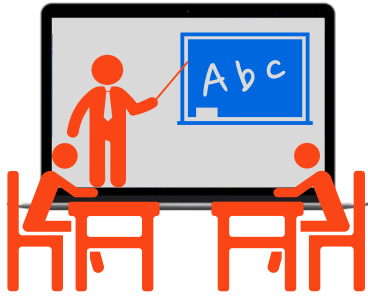
Demonstrate the use of the If statement by building a workflow that tells whether the user has passed the exam or not.

- Insert an **Input Dialog** activity in the designer panel.
- Enter the text “**Name**” in the *Title*, and “**Enter Your Name**” in the *Label*.
- Define a variable called **name** for this activity with Variable Type as **string** using **Variables** panel.
- Enter this variable in the **Result** property of this **Input Dialog** activity using its **Properties** panel.
- Insert another **Input Dialog** activity below the first Input Dialog activity.
- Enter the text “**Marks**” in the *Title*, and “**Enter your Marks in numeric value**” in the *Label*.
- Define a variable called **intMarks** for this activity with Variable Type as **Int32** using the **Variables** panel.
- Enter this variable in the **Result** property of this **Input Dialog** activity using its **Properties** panel.
- Insert **If** activity below the second Input Dialog activity.
- Enter condition **intMarks >= 60**.
- Insert two **Message Box** activities. One in **Then** box and another in **Else** box.
- In the first Message Box activity, enter the text “**Congratulations!**” + **name** + “**, you have passed the exam.**”
- In the second Message activity, enter the text “**Hello!**” + **name** + “**, you have failed the exam.**”
- Run the process

- Enter the name of a student in the Name box, say **John**, and **75** in the marks box. Passed result displays.
- Run the program again using marks **50** for John...he has failed.

## Slide 14

### Practice Exercise



- Build a workflow using an If activity that tells whether the user will get second Marshmallow or not.
- Ask the user "Do you want to eat your first Marshmallow now or after 5 minutes?"
- If the user answers "Now", respond with "Oops! You will not get the second Marshmallow."
- If the user answers "After 5 minutes", respond with "Congrats! You will also get the second Marshmallow."
- If the answer is other than "Now" or "After 5 minutes", respond with "Invalid Input".

Build a workflow using an If activity that tells whether the user will get a second Marshmallow or not.

- Ask the user "Do you want to eat your first Marshmallow now or after 5 minutes?"
- If the user answers "Now", respond with "Oops! You will not get the second Marshmallow."
- If the user answers "After 5 minutes", respond with "Congrats! You will also get the second Marshmallow."
- If the answer is other than "Now" or "After 5 minutes", respond with "Invalid Input".

### Process Overview

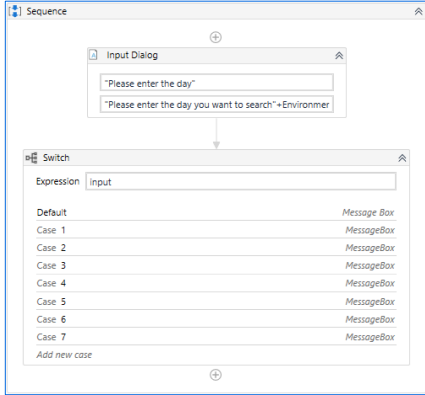
- START
- Use an Input Dialog activity to ask the user "Do you want to eat your first Marshmallow now or after 5 minutes?"
- Store user response in a string variable.
- Use an If activity to check the user response
  - If the answer is "Now", use a Message Box activity to display "Oops! You will not get a second Marshmallow."
  - If the answer is "After 5 minutes", use a Message Box activity to display "Congrats! You will also get a second Marshmallow."
  - If the answer is other than "Now" or "After 5 minutes", use a Message Box activity to display "Invalid Input".
- STOP



## Slide 15

### Switch Activity

Executes a set of statements out of multiple, based on the value of a specific expression.



A Switch activity executes a set of statements out of multiple, based on the value of a specific expression. It is used in place of an If activity when at least 3 potential courses of action are needed. It can be useful to categorize data according to a custom number of cases

Here, the condition does not hold a Boolean value (like in the case of an If statement), but multiple values. By default, the Switch activity uses the integer argument, but can be changed from the Properties panel, from the TypeArgument list.


A Switch activity can be used to store data into multiple spreadsheets or sort through names of employees.

**Example:**

A week has seven days. Each of the seven days can be associated with a number (1-7 for Monday-Sunday). To display the name of the day associated with a particular number, 7 cases are built using the Switch activity.

## Slide 16

### Classroom Exercise



Demonstrate how to use a **Switch** activity by building a workflow that does the following:

- Takes numbers between 1 and 7 as input from the user. Here, 1 is for Monday, 2 for Tuesday, and so on till Sunday.
- Checks the input number against the defined condition:
  - If the number entered is between 1 and 5, then it displays on the screen "It's a weekday" in a message box.
  - If the number entered is 6 or 7, then it displays on the screen "It's weekend" in a message box.
  - If the number entered is not between 1 and 7 then it displays an error saying, "Invalid entry, please enter the number between 1 and 7 only".

Demonstrate how to use a Switch activity in workflows.

How to use a Switch activity in the workflow to ask the day number from the user and tell her whether it is a weekday or a weekend.

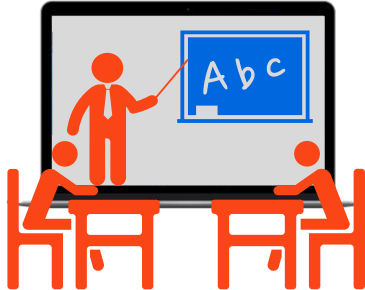
- Insert an **Input Dialog** activity in the designer panel.
- Enter the text "**Taking user input**" in the *Title*.
- In the *Label* enter "**Please enter the number of your choice**" + Environment.NewLine + "**1. Monday**" + Environment.NewLine + "**2. Tuesday**" + Environment.NewLine + "**3. Wednesday**" + Environment.NewLine + "**4. Thursday**" + Environment.NewLine + "**5. Friday**" + Environment.NewLine + "**6. Saturday**" + Environment.NewLine + "**7. Sunday**".
- Create a variable **intDayNumber** for *Expression*.
- Insert a **Message Box** activity in the *Default* section of the Switch activity and enter the text "Invalid entry, please enter the number between 1 and 7 only".
- Create 7 cases, one for each day. Enter **1** in the first case. Insert a **Message Box** activity in the *Activity Area* and enter the text "It's a weekday".
- Repeat the same process for the remaining 6 cases...2 for Tuesday, 3 for Wednesday, 4 for Thursday, 5 for Friday, 6 for Saturday, and 7 for Sunday
- For Saturday and Sunday, insert "It's weekend" in their message boxes.
- Run the program and test few variations.

**Outcome:** Program runs successfully.

## Slide 17

### Practice Exercise

---



Build a workflow using **Switch** activity that asks **users** their eye color, and display their personality in a message box.

- Ask the user for their eye color.
- If the user enters "Blue", respond with "You must be very Brave!"
- If the user enters "Green", respond with "You must be Generous!"
- If the user enters "Gray", respond with "You must be very Wise!"
- If the user enters "Black", respond with "You must be very Bold!"

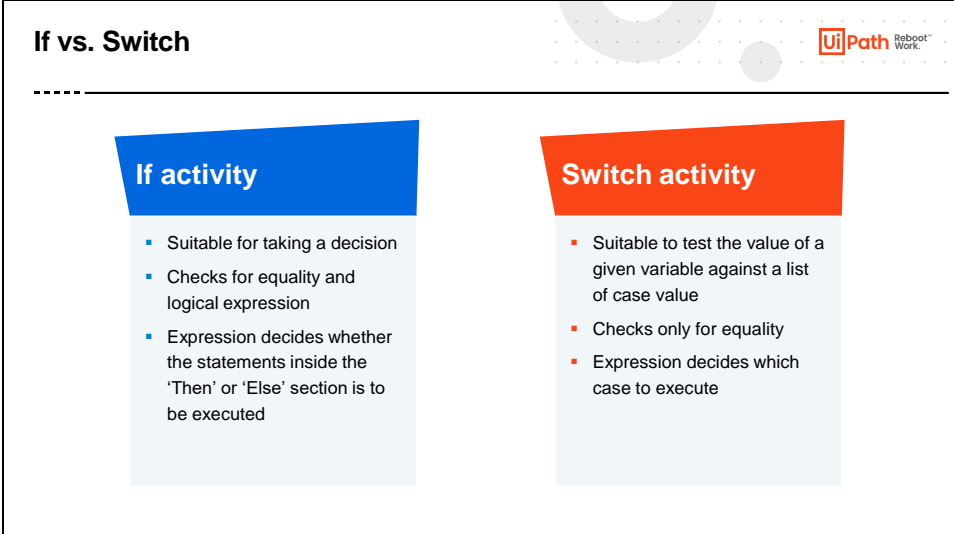
Build a workflow **Switch activity** that asks users their eye color, and display their personality in a message box.

- Ask the user for their eye color.
- If the user enters "Blue", respond with "You must be very Brave!"
- If the user enters "Green", respond with "You must be Generous!"
- If the user enters "Gray", respond with "You must be very Wise!"
- If the user enters "Black", respond with "You must be very Bold!"

### Process Overview

- START
- Use an Input Method activity to get the eye color input of the user.
- Use a Switch activity to compare input with four different cases – Blue, Green, Gray, and Black.
- Use Message Box activities to display result of each case
  - For "Blue", respond with "You must be very Brave!"
  - For "Green", respond with "You must be Generous!"
  - For "Gray", respond with "You must be very Wise!"
  - For "Black", respond with "You must be very Bold!"
- STOP

## Slide 18



**If vs. Switch**

-----

If activity	Switch activity
<ul style="list-style-type: none"><li>▪ Suitable for taking a decision</li><li>▪ Checks for equality and logical expression</li><li>▪ Expression decides whether the statements inside the 'Then' or 'Else' section is to be executed</li></ul>	<ul style="list-style-type: none"><li>▪ Suitable to test the value of a given variable against a list of case value</li><li>▪ Checks only for equality</li><li>▪ Expression decides which case to execute</li></ul>

If and Switch, both are used for decision control. However, there are few differences between them.

**If activity:**

- Suitable for taking a decision
- Checks for equality and logical expression
- Expression decides whether the statements inside the 'Then' or 'Else' section is to be executed

**Switch activity:**


- Suitable to test the value of a given variable against a list of case value
- Checks only for equality
- Expression decides which case to execute


The use of the Switch activity is preferred over If activity because it is easier to debug and read.

Slide 19

## Loops

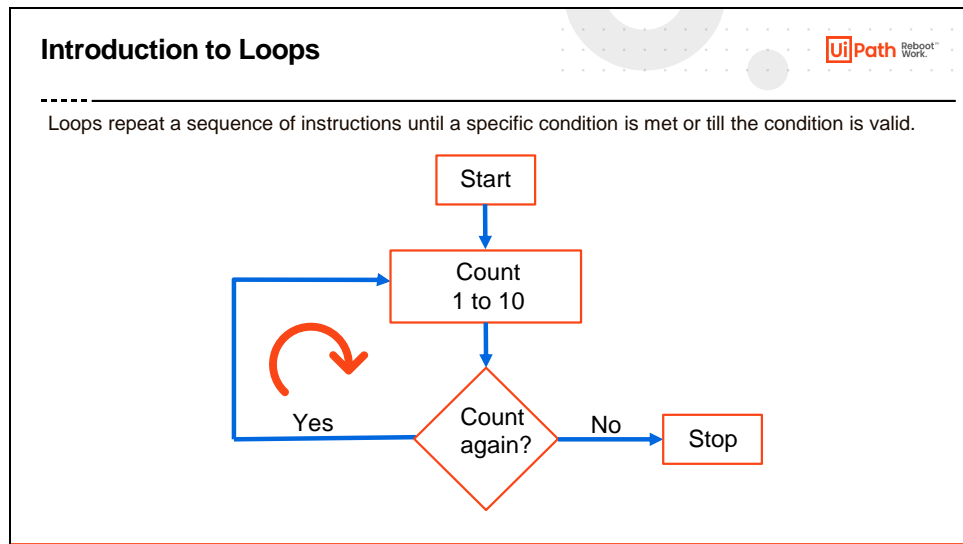
- Introduction to Loops
- Types of Loops
  - Do While
  - While
  - For Each





This section explains loops and its several types in detail.

## Slide 20

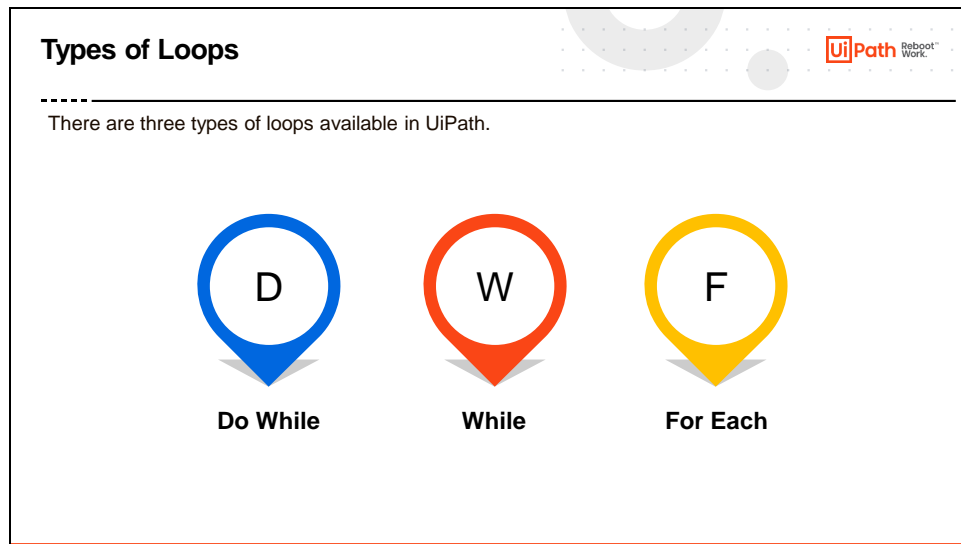


A loop is a programming structure that repeats a sequence of instructions until a specific condition is met or till the condition is valid.

**Example:** Suppose the user wants to create a program that counts from 1 to 10. The user may want to restart the count once it is complete. This is a loop. The user can continue doing so as long as desired. At the instance when the user doesn't want to count again, he breaks out of the loop.

Here, "Count 1 to 10" is the instruction, "count again" is the condition. As long as the condition is met the count continues.

## Slide 21



There are three types of loops in UiPath:

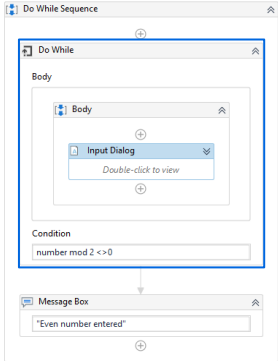
- **Do While:** Executes the contained activities first and then checks if the condition is true. This also means that a Do While loop will be executed at least once even if the condition is not true.
- **While:** Executes the contained activities while the provided condition is true. It checks the condition at the beginning before getting into the loop.
- **For Each:** Lets the user iterate through the elements of lists, datasets, etc., and perform an action on each element individually.

These are discussed in detail in the subsequent slides.

## Slide 22

## Do While

Executes a specific sequence while a condition is met. The sequence is executed at least once, and then until the specified condition is no longer met. The condition is evaluated after each execution of the sequence.



The Do While activity enables the user to create a loop that executes a specified part of automation while a condition is met. The sequence is executed at least once, and the condition is evaluated before each execution of the sequence. When the specified condition is no longer met, the program exits the loop.

This type of activity can be useful to step through all the elements of an array or execute a particular activity multiple times. The user can increment counters to browse through array indices or step through a list of items.


For example, a program to input an even number. The Do while loop keeps asking for an input until an even number is entered.

Break activity can be used to exit the loop at any desired point (even when the specified condition is valid).



## Slide 23

### Classroom Exercise



Demonstrate how to use **Do While loop** by building a workflow which asks for an even number. The program continues asking for input till an even number is entered.

- Ask for an even number input from the user.
- If the user enters an odd number:
  - Display "It's an odd number. Try again."
  - Again, ask for an even number input.
- If the user enters an even number:
  - Display "Right! It's an even number."
  - End the program.

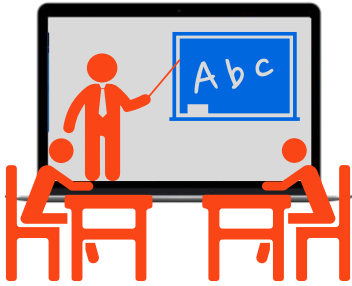
Demonstrate how to use a Do While loop by building a workflow that asks for an even number. The program continues asking for input till an even number is entered.

- Insert a **Do While** activity in the designer panel.
  - The **Do While** activity will run the loop and execute the statement depending on the specified condition.
- Insert an **Input Dialog** activity in the Body section.
- Enter text "**Even Number Identifier**" in the *Title* and "**Enter an even number**" in the *Label*.
- Define a variable called **intNumber** for this activity using the **Variables** panel. Set Variable Type as **Int32**.
- Enter this variable in the **Result** property of this activity using its **Properties** panel.
- Insert an **If** activity below the Input Dialog activity. In the Condition box, enter **intNumber mod 2=0**.
- In the Then section, insert a Message Box activity. Enter text "Right! It's an even number."
- In the Else section, insert another Message Box activity. Enter text "It's an odd number. Try again."
- In the condition box of the While activity, enter **intNumber mod 2 <> 0**.
- Run the process.
- Try 7
  - Result is an odd number.
- Try 9
  - Result is an odd number.
- Try 4
  - Result is an even number.

**Outcome:** Program runs successfully.

## Slide 24

### Practice Exercise



- Build a workflow using a Do While loop for creating a 'Guessing Game' with the following conditions:
- Generate a random number and prompt the user to input a number.
- If the input number is greater than the number generated, then it should display the message: 'Please enter a lesser number'.
- If the input number is lesser than the number generated, then it should display the message: 'Please enter a greater number'.
- The loop keeps on running until the input number equals to the generated number.

Build a workflow using a **Do While** loop creating a 'Guessing Game' with the following conditions:

- Generate a random number and prompt the user to input a number.
- If the input number is greater than the number generated, then it should display the message: 'Please enter a lesser number'.
- If the input number is lesser than the number generated, then it should display the message: 'Please enter a greater number'.
- The loop keeps on running until the input number equals to the generated number.

### Process Overview

- START
- Use an Input Dialog activity within a Do While activity to get the guessed number from the user.
- For Do While activity, set the condition to check guessed number is not equal to the actual number.
- Use a Message Box activity to display “You Guessed it correct” for the correct match.
- Use an If activity within the Do While loop to check if the guessed number is equal to the actual number.
  - If correct, use a Message Box activity to display “You Guessed it correct” for the correct match.
  - Use another If activity within the Else section to check if the guessed number is greater than the actual number.
    - If correct, use a Message Box activity to display “Please try a smaller number”.

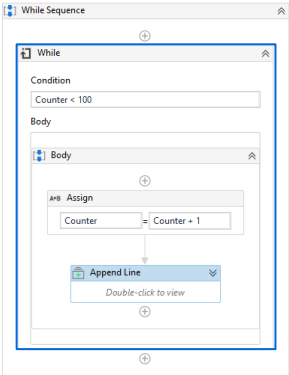
- If incorrect, use a Message Box activity to display “Please try a greater number”.
- STOP

## Slide 25

## While

-----

Executes a specific sequence while a condition is met. The condition is evaluated before each execution of the sequence.



The While activity enables the user to create a loop that executes a specific process repeatedly, while a specific condition is met. The condition is evaluated before each execution of the sequence.

The main difference between While and the Do While activity is that, in While, the condition is evaluated before the body of the loop is executed.


This type of activity can be useful to step through all the elements of an array or execute a particular activity multiple times. The user can increment counters to browse through array indices or step through a list of items.

For example, a program to print a list of prime numbers up to a given number.


Break activity can be used to exit the loop at any desired point (even when the specified condition is valid).

## Slide 26

## Classroom Exercise



---



Demonstrate how to use a **While loop** to print a list of numbers.

- Ask the user to input a number greater than 1.
- Print a list of prime numbers between 1 and the number given by the user.

Demonstrate how to use a **While** loop.


How to print a list of numbers starting from 1 till the number you want using a **While** activity.

- Insert an **Input Dialog** activity in the designer panel.
- Enter the text “Enter Number” in the *Title* and “Enter any number you want to print” in *Label*.
- Define a variable called **intNumber** for this activity using the **Variables** panel. Set Variable Type as **Int32**.
- Enter this variable in the **Result** property of this activity using its **Properties** panel.
- Insert an **Assign** activity below the Input Dialog activity.
- In *To* text box, press **Ctrl+K** and enter **intTemp** as a variable, and **1** in the adjacent box. In the Variables panel, set Variable Type as **Int32** for **intTemp**.
- Insert a **While** activity below the Assign activity.
- In the **Condition** box of While activity enter **intTemp<=intNumber**.
- In the **Body** section of the While activity, insert a **Write Line** activity.
- Enter the expression **intTemp.ToString** in the text box.
- Insert an **Assign** activity below the **Write Line** activity.
- In the *To* text box enter the variable **intTemp**, and enter **intTemp+1** in the adjacent text box.
- Run the program
- Enter **15** in the Input box, and click **OK**.
- Go to **UiPath** studio to check the **Output** panel for the result

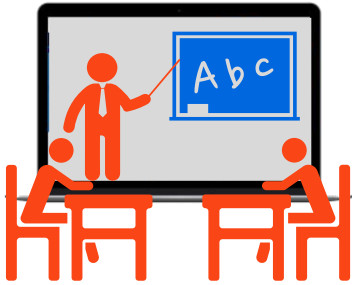
**Outcome:** Numbers are printed from 1 to 15 Program ran successfully.

## Slide 27

### Practice Exercise



---



Build a workflow using a While loop which tells the user if the input is a prime number or not.

- Ask the user to input a number.
- Check if it is a prime number.
- If the input number is prime, then display "It is a prime number" in a message box.
- If the input number is not prime, then display "It is not a prime number" in a message box.

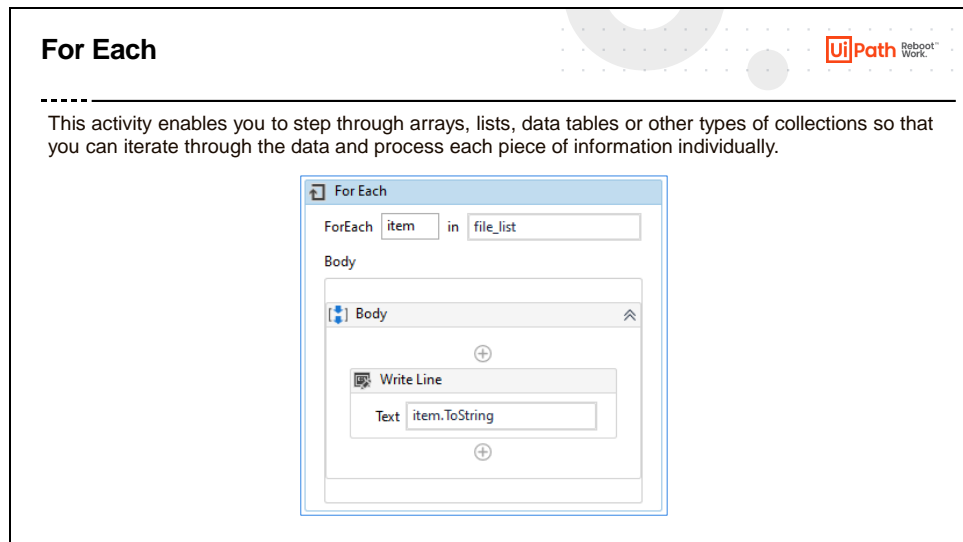
Build a workflow using a While loop which tells the user if the input is a prime number or not.

- Ask the user to input a number.
- Check if it is a prime number.
- If the input number is prime, then display "It is a prime number" in a message box.
- If the input number is not prime, then display "It is not a prime number" in a message box.

### Process Overview

- START
- Use an Input Dialog activity and ask for any number from the user and store in a variable called **intNumber**.
- Create two more variables **intRandom** and **c** with Variable Type as **Int32** and Default value as **2** and **0** respectively in the variables panel.
- Use a While activity and set the condition to **intRandom < Number**.
- Use an If activity within the While activity and set the condition to **intNumber mod intRandom = 0**.
- Use an Assign activity within the **Then** section and increment value of **intCount** by **1**.
- Use an Assign activity after/below the If activity, and increment value of **intRandom** by **1**.
- Use another If activity after/below the While activity and enter condition **intCount > 0**.
- Use a Message Box activity within the Then section to display "It is not a prime number".
- Use a Message Box activity within the Else section to display "It is a prime number".
- STOP

## Slide 28



It performs an activity or a series of activities on each element of a collection.

The For Each activity enables the user to step through arrays, lists, data tables or other types of collections, so that the user can iterate through the data and process each piece of information individually.

This is very useful in the data processing. Consider an Array of integers. For Each would enable the robot to check whether each numeric item fulfills a certain condition.


The screenshot on the slide depicts the structure of the For Each loop.

A Break activity can be used to exit the loop at any desired point (even when the specified condition is valid).




## Slide 29

## Classroom Exercise



---



Demonstrate how to use a **For Each** activity. Build a workflow that displays the directory name of all the files from a folder.

- Locate and select a folder containing multiple files.
- List the directory path of all the files in the Output panel.

Demonstrate how to use a **For Each** activity.

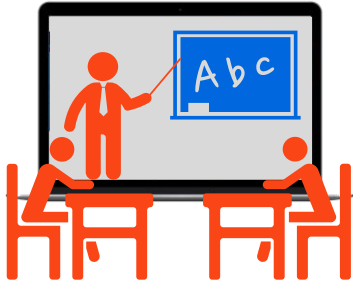
How to use a For Each activity in UiPath Studio and display directory name of all the files from a folder in the Output panel.

- Insert a **Select Folder** activity in the designer panel.
- Define a variable **folderName** in the **Variables** panel and assign it using the **Output** property in the **Properties** panel.
- Insert an **Assign** activity below the **Select Folder** activity.
- In the *To* box, press **Ctrl+K** and enter **fileList** as another variable. In the adjacent box enter **Directory.GetFiles(folderName)**.
- In the **Variables** panel, change its **Variable Type** to array of strings.
- Insert a **For each** activity below the **Assign** activity.
- Insert the text **item** in the first text box and the variable **fileList** in the second text box.
- Insert a **Write Line** activity in the body section of the **For Each** activity.
- Enter the text **item.ToString** in it.
- Run the file.
- From the window select a folder containing some files.
- Go to the Output panel to see result

Outcome: Output panel lists all file names Program runs successfully.

## Slide 30

### Practice Exercise



- Build a workflow to display file names from a folder in the Output panel and also store names in an MS Word file..
- Locate and select a folder containing multiple files.
- List the directory path of all the files in the Output panel.
- Also, store the updated names in an MS Word file and save and close it.

Build a workflow using a **For Each** activity which performs the following:

- Locate and select a folder containing multiple files.
- List the directory path of all the files in the Output panel.
- Also, store the updated names in an MS Word file and save and close it.



### Process Overview

- START
- Use a Select Folder activity to select a folder containing a few files.
- Use an Assign activity to store file names in an array.
- Use an Attach Window activity below the Assign activity and select MS Word window.
- Use a For Each activity to iterate through each file name in the array.
- Use a Write Line activity within the For Each activity to display file names in the Output panel.
- Use a Type Into activity below the Write Line activity to store file names in an MS Word file.
- Use Click and Send Hotkey activities to save and close the file.
- STOP

Slide 31

## Other Control Flow Activities

- Delay
- Break
- Assign
- Continue
- Parallel




This section gives an overview of the other control flow activities in UiPath.

## Slide 32

### Other Control Flow Activities

-----

In addition to decision-based and iteration-based control flow activities, there are other activities in UiPath that help in controlling the flow of the program. These are:



Activity	Icon Description
Delay	Blue chevron with a pause symbol (two vertical bars).
Break	Orange chevron with a break symbol (a circle with a diagonal slash).
Assign	Yellow chevron with an assign symbol (a square with a diagonal line and a small square).
Continue	Grey chevron with a play symbol (a right-pointing triangle).
Parallel	Pink chevron with a parallel symbol (three vertical bars of increasing height).

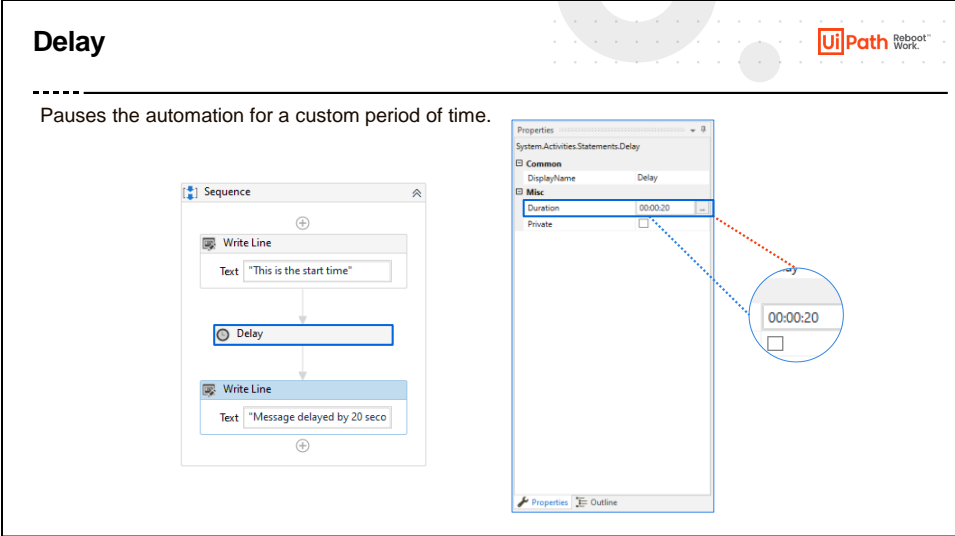
As discussed earlier, in addition to decision-based and iteration-based control flow activities, there are other activities in UiPath that help in controlling the flow of the program. These are:

- Delay
- Break
- Assign
- Continue
- Parallel

## Slide 33

## Delay

Pauses the automation for a custom period of time.



The screenshot displays the UiPath IDE interface. On the left, a 'Sequence' container holds three activities: a 'Write Line' activity with the text 'This is the start time', followed by a 'Delay' activity, and another 'Write Line' activity with the text 'Message delayed by 20 seco'. The 'Delay' activity is highlighted with a blue border. On the right, the 'Properties' panel is open for the 'Delay' activity. The 'Duration' property is set to '00:00:20'. A red dashed line connects this property to a circular callout on the right, which shows a digital clock display set to '00:00:20'.


The Delay activity enables the user to pause the automation for a custom period of time. The duration can be set from the Properties panel under the Duration tab in hh:mm:ss format.

This activity is useful in projects that require good timing, such as waiting for a specific application to start or waiting for some information to be processed so that it can be used in another activity.


For example, the sequence is delayed by 20 seconds after the text 'This is the start time' is displayed.

## Slide 34

## Classroom Exercise



---



Demonstrate how to use a **Delay** activity to delay a process by a certain amount of time.

- Ask the user to open a notepad within 30 seconds.
- Pause the process for 30 seconds.
- Write a piece of text in the opened notepad.

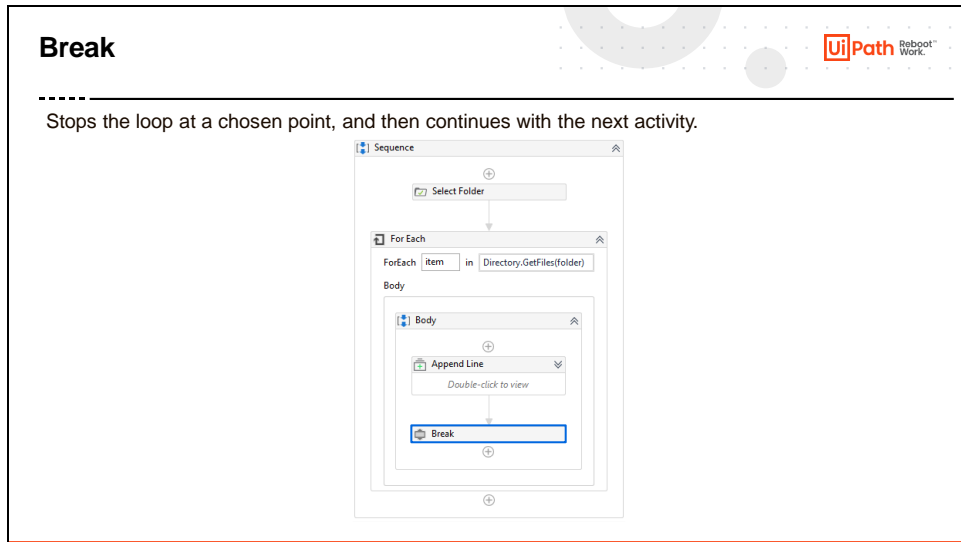
Demonstrate how to use a **Delay** activity to delay a process by a certain amount of time.

Create a workflow that tells user to open a notepad, and then it will wait for 30 seconds for user to open the notepad. And then it writes a text in the opened notepad.

- Open a notepad on the desktop.
- In UiPath Studio, insert a **Message Box** activity in the designer panel.
- Enter the text “Hi, open a new notepad within 30 seconds.”
- Insert a Delay activity below the Message Box activity.
- Go to the Properties panel of the Delay activity and set duration as 30 seconds.
- Insert a Type Into activity below the Delay activity.
- Click the “Indicate on screen” link and indicate the editor area of the already open notepad. Enter the text “Automation is future.”
- Close already opened notepad.
- Go to Studio and run the program.
- Process tells user to open a notepad. Click OK. Now the 30 seconds timer has started and running in the background.
- Open a notepad and wait for the timer to run out.

**Outcome:** Text gets written in the notepad after some delay. This is how the Delay activity works. It makes the process wait for certain amount of time, and then continues running.

## Slide 35




The Break activity enables the user to stop the loop at a chosen point, and then continues with the next activity.

For example: Using a Break activity, the user exits the For Each activity and continues the workflow with the activity that follows it.

## Slide 36

## Classroom Exercise

---



Demonstrate how to use a **Break** activity to stop a process by building a workflow that does the following:

- Run a loop using the For Each activity to append a text in each item in an array of integers.
- After each iteration, display a message box and ask the user "One iteration is done. Do you want to stop?"
- If the user replies with 'Yes' then the iteration stops, else it continues.
- Display the output in the Output panel.

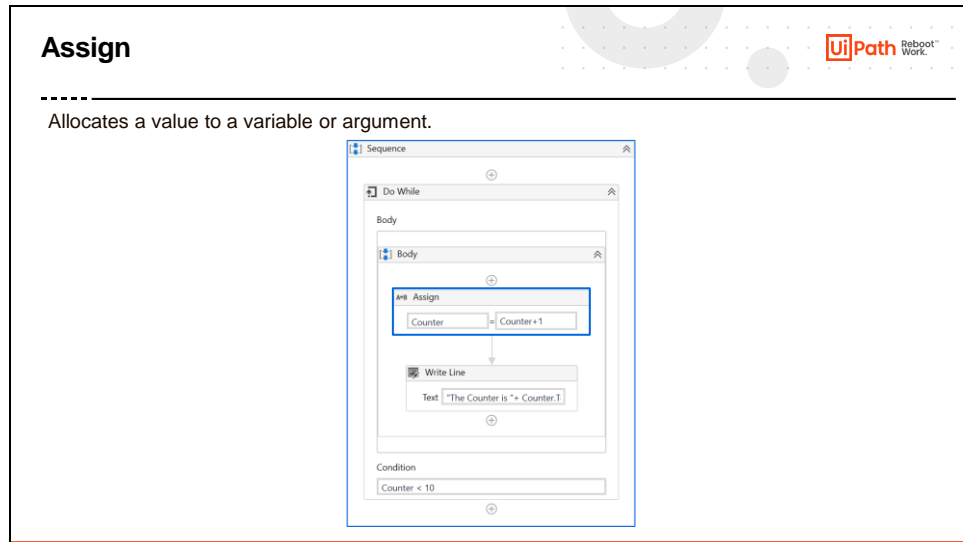
Demonstrate how to use a **Break** activity to stop a process.

Run a loop using For Each activity to append a text in each item in an array of integers and display the result in the Output panel. Stop the iteration whenever user wants to stop.

- Insert an **Assign** activity in the designer panel.
- In the *To* box, press **Ctrl+K** and enter **newArray** as another variable.
- In the adjacent box enter four integers **{45,56,67,78}**.
- In the **Variables** panel, change the **Variable Type** of **newArray** to array of integers.
- Insert a **For Each** activity below the Message Box activity. Leave text **item** in the first box as it is. In the second text box enter variable **newArray**.
- In the Body section enter a **Message Box** activity. Enter text **item.ToString + " is a number."**.
- Insert an **Input Dialog** activity below the **Message box** activity.
- Enter the text "Question" in the *Title* and "One iteration is done. Do you want to stop?" in *Label*.
- Define a variable called **userReply** for this activity using the **Variables** panel. Set Variable Type as **string**.
- Enter this variable in the **Result** property of this activity using its **Properties** panel.
- Insert an **If** activity below the Input Dialog activity. Enter the condition **userReply= "Yes"**. Insert a **Break** activity in the **Then** box.
- Run the program with few variations.



## Slide 37



The Assign activity allocates any value to a variable or argument.

As a control flow activity, it can be used to:


- Increment the value of a variable in a loop.
- Sum up the value of two or more variables and assign the result to a different variable.
- Assign values to an array.

For example, the Assign activity within the loop increases the value of the variable Counter by 1 at each iteration.

## Slide 38

## Classroom Exercise

---



Demonstrate the use of **Assign** activity by building a workflow in which we assign a value to a variable.

- Ask the user to input his name and then assign it to a string variable using Assign activity
- Display "Hello! How can I assist you, User". Replace "User" with user's name.

Demonstrate the use of **Assign** activity to assign a value to a variable.

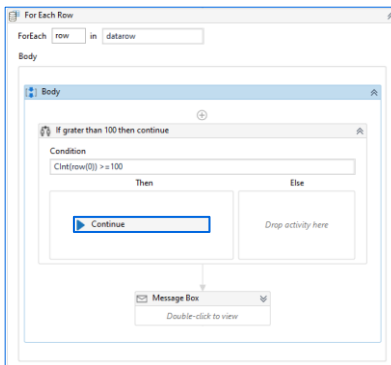
- Ask user to input her name and assign it to a string variable using Assign activity. And then display her name combined with an additional text in a message box.
- Insert **Input Dialog** activity in the designer panel.
- Enter the text “Question Box” in the *Title* and “What’s your name?” in *Label*.
- Define a variable called **name** 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 **Assign** activity below Input Dialog activity. Press **Ctrl+K** and enter the text **message** as variable in the *To* text box, and “Hello! How can I assist you, “ + **name** in the adjacent box.
- Insert **Message Box** activity below **Assign** activity and enter variable **message** in the text box.
- Run the file.
- Enter a name say, Andrew, and click OK.

**Outcome:** Result received as expected.

## Slide 39

## Continue

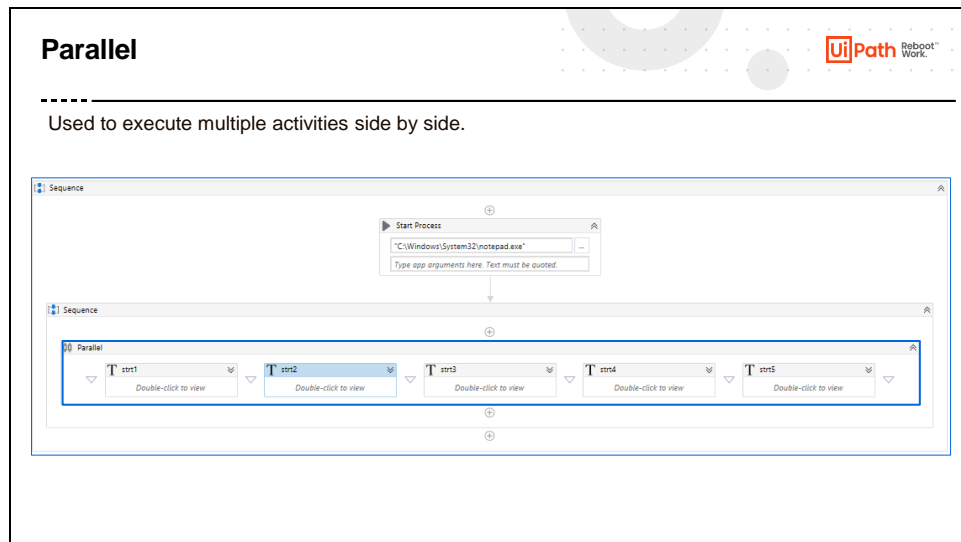
Enables you to skip the remaining steps in the current iteration inside a loop activity, such as For Each, While, or Do While loop.



Enables you to skip the remaining steps in the current iteration inside a loop activity, such as For Each, While, or Do While loop and continues the execution with the next iteration.

For example: Continue activity used with a condition (using the If activity) inside a For Each loop.

## Slide 40



The Parallel activity executes child activities in parallel. This means that by using Parallel, multiple activities can be executed side by side. It is helpful when there is the need to run several processes at the same time.

A Parallel activity lets the user schedule two or more child activity branches for processing simultaneously. However, it cannot run multiple processes in the foreground at the same time. In order to run more than one process at the same time, the user has to choose a single activity branch to run in the foreground, and the remaining activity branches have to run in the background. To do this, `SendWindowMessages` and/or `SimulateType` property of the activities are used.


If `SendWindowMessages` or `SimulateType` is not used, then the Parallel activity begins processing with the execution of one activity branch at a time. It completes one activity branch, and then randomly picks another activity branch. It gives the result randomly when a user allows this process in the parent activity.

For example: There are 5 `TypeInto` activities (numbered 1 to 5) placed in the Parallel activity. On execution, the activity gives the results randomly like 34512, 43125, 53142, 32541, etc.

## Slide 41

## Classroom Exercise

---



Demonstrate how to run two processes in parallel using a **Parallel** activity by writing a piece of text in two separate notepad windows.

- Create two processes to write texts in two different notepads.
- Write texts in both the notepads simultaneously.

Demonstrate how to run two processes in parallel using a **Parallel** activity by writing a piece of text in two separate notepads.

- Create two notepads with names “foreground.txt”, and “background.txt”, and open them side by side on the desktop.
- In the UiPath Studio, insert a **Parallel** activity in the designer panel, and insert two **Sequence** activities in it.
- Insert a **Type Into** activities in both the **Sequence** activities.
- Click the “Indicate on screen” link of the first **TypeInto** activity and indicate editor area of foreground.txt file.
- Enter 5 sentences in the text box of the **Type Into** activity.
- Click the “Indicate on screen” link of the second **TypeInto** activity and indicate editor area of background.txt file.
- Enter different 5 sentences in the text box of the second **TypeInto** activity.
- Run the process.

**Outcome:** Text gets written in the foreground.txt file first. Only then it gets written in the background.txt. Both Type Into activities are not running in parallel because they are running in foreground.


Multiple processes can run in parallel, but only in background. Only one process can run in foreground. Run the second **Type Into** activity in the background by using the **SendWindowMessages** property.

- Click the **Type Into** activity container in the second **Sequence** activity.
- In its Properties panel, set **SendWindowMessages** to **True**.
- Empty both the notepads.
- Run the process again

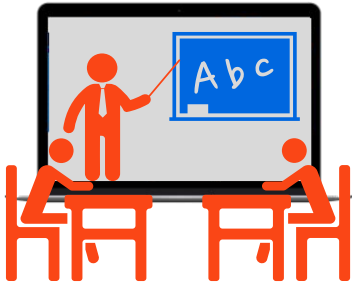
**Outcome:** Text gets written in both the Notepads at the same time. Parallel processes are running successfully.

## Slide 42

## Practice Exercise



---



Build a workflow using a **Parallel** activity to do the following:

- Perform following activities in parallel:
  - Search UiPath website on Google, copy about UiPath from the search result.
  - Search for "What is automation" on Google, copy the definition from the search result.
  - Search for "Automation future" on Google, copy the first search result.
- Finally, store all copied text in an MS Word file.
- Save and close the MS Word file.

Build a workflow using a Parallel activity to do the following:

- Perform the following activities in parallel:
  - Search UiPath website on Google, copy about UiPath from the search result.
  - Search for "What is automation" on Google, copy the definition from the search result.
  - Search for "Automation future" on Google, copy the first search result.
- Finally, store all copied text in an MS Word file.
- Save and close the MS Word file.

### Process Overview

- START
- Use three Sequence activities in parallel within the Parallel activity.
- In the first Sequence activity:
  - Use an Open Browser activity to open the Google website "www.google.com"
  - Use a Type Into activity to search for UiPath website.
  - Set the SendWindowMessages property to True for this Type Into activity.
  - Copy about UiPath from the search result using the Get Text activity.
- In the second Sequence activity:
  - Use an Open Browser activity to open Google website "www.google.com"
  - Use a Type Into activity to search "What is automation".
  - Set the SendWindowMessages property to True of this Type Into activity.
  - Copy the search result using the Get Text activity.
- In the third Sequence activity:

- Use an Open Browser activity to open Google website “www.google.com”
  - Use a Type Into activity to search for “Automation is Future”.
  - Set the SendWindowMessages property to True of this Type Into activity
  - Copy the search result using the Get Text activity.
- Use an Attach window and a Type Into activity to insert text from all three parallel Sequence activities into an MS Word file.
- Use Send Hotkey activities to Save and close the MS Word file.
- STOP



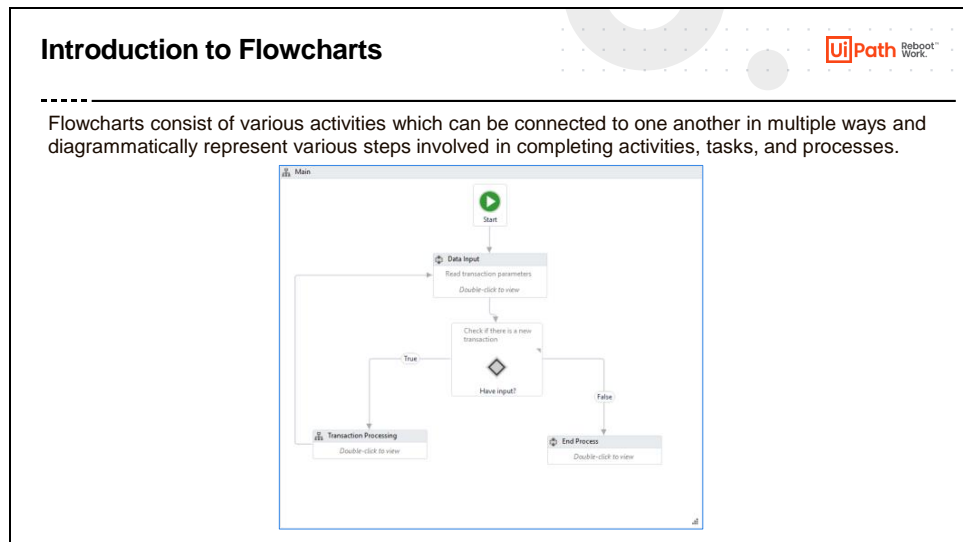
Slide 43

## Flowcharts

- Introduction to flowcharts
- Decision making in flowcharts
- Loops in flowcharts
- Nesting flowcharts and sequences
- Sequences vs flowcharts

This section explains flowcharts in detail.

## Slide 44



A flowchart is a type of project that consists of various activities which can be connected to one another in multiple ways. It is a diagrammatical approach that represents various steps involved in completing activities, tasks, and processes. This helps the user to easily view and follow the process.


Flowcharts can be either used as stand-alone automation projects or included in procedures of larger projects.

The most important aspect of flowcharts is that, unlike sequences, they present multiple branching logical operators, that help to create complex business processes and connect activities in multiple ways.

## Slide 45

## Classroom Exercise

---



Demonstrate how to use **Flowcharts** by building a workflow that tells the user whether he has passed the exam or not.

- Take input from a user of name & marks obtained in an exam.
- Check if the marks are greater than 60 or not.
- Display the result in a message box for Pass as "Congratulations! you have passed the exam."
- Display the result in a message box for Fail as "Hello User, you have failed the exam." Replace "User" with the user's name.

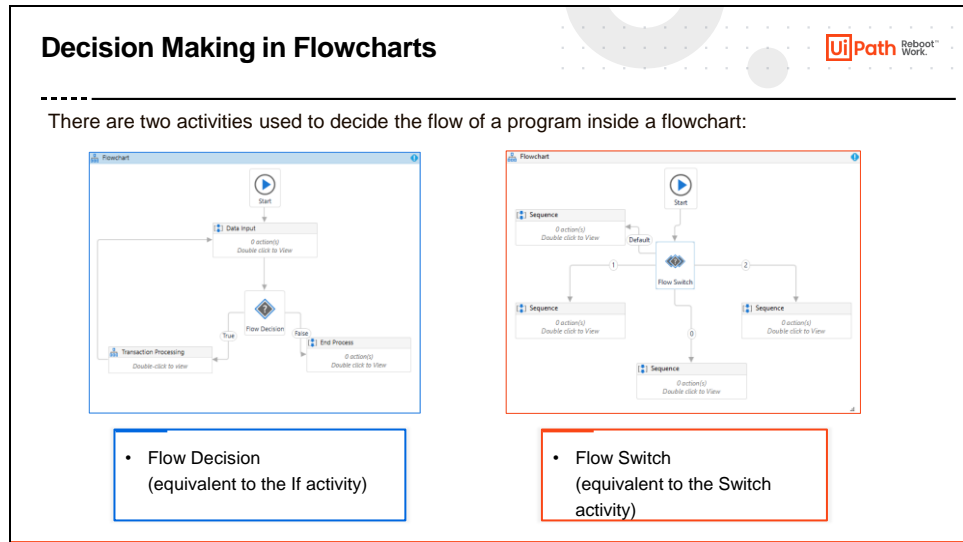
Demonstrate how to use **Flowcharts** by building a code that tells the user whether he has passed the exam or not. Ask for his name and marks obtained, and then display the exam outcome in a message box.

- Insert a Flowchart activity in the designer panel.
- Insert an **Input Dialog** activity below the Start node in the Flowchart container.
- Enter the text "**Name**" in the *Title*, and "**Enter Student Name**" in the *Label*.
- Define a variable called **Names** for this activity using the **Variables** panel. Set Variable Type as **string**.
- Enter this variable in the **Result** property of this activity using its **Properties** panel.
- Insert another **Input Dialog** activity below the first Input Dialog activity.
- Enter the text "**Marks**" in the *Title*, and "**Enter your Marks in numeric**" in the *Label*.
- Define a variable called **intMarks** for this activity using the **Variables** panel. Set Variable Type as **Int32**.
- Enter this variable in the **Result** property of this activity using its **Properties** panel.
- Insert a **Flow Decision** activity below the second Input Dialog activity.
- Enter the condition **intMarks >= 60** in its **Properties** panel.
- Insert two **Message Box** activities. One on the left and another on the right of the Flow Decision activity.
- In the first Message Box activity, enter the text "**Congratulations!**" + **Names** + ", you have passed the exam."
- In the second Message activity, enter the text "**Hello**" + **Names** + ", you have failed the exam."
- Run the process

- Enter name of a student in the Name box, say **Ron**, and **75** in the marks box Result shows he has passed.
- Again, run the program using marks **50** for Ron. Result shows he has failed.

**Outcome:** Program runs successfully.

## Slide 46



There are two activities used to decide the flow of a program inside flowcharts. These are:

- **Flow Decision:** It is an activity that executes one of two branches, depending on whether a specified condition is met. The branches are titled **True** and **False** by default, but their names can be changed in the **Properties** panel. This activity can only be used in a **Flowchart** and is equivalent to the **If** activity.


Important properties of this activity are:

- **Condition:** The condition to be analyzed before one of the two branches is executed. This field supports only Boolean expressions.
  - **TrueLabel:** Enables the user to provide a description of the case in which the condition is met. The description can be viewed by hovering the cursor over the Flow Decision activity. By default, this is filled in with True.
  - **FalseLabel:** Enables the user to provide a description of the case in which the condition is not met. The description can be viewed by hovering the cursor over the Flow Decision activity. By default, this is filled in with False.
- **Flow Switch:** It is an activity that splits the control flow into three or more branches, out of which a single one is executed based on a specified condition. This activity can only be used in a **Flowchart** and is equivalent to the **Switch** activity.

## Slide 47

### Classroom Exercise

---



Demonstrate how to make **decisions** in Flowcharts by building a workflow that tells the user if he is eligible to vote or not.

- Ask the user to input his age.
- Voting age should be greater than or equal to 18.
- Display "You can vote!" in a message box if the input age is greater than 18.
- Display "You cannot vote!" in a message box if the input age is less than 18.
- Use a Flow Decision activity to decide whether the user can vote or not.

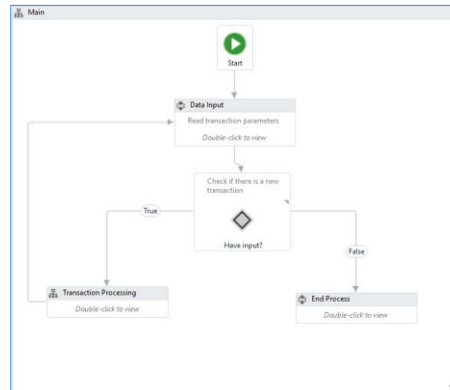
Demonstrate how to make decisions in Flowchart. Ask the user to input their age, and display if they can vote in a message box. Use a Flow Decision activity to decide whether they can vote.

- Insert a Flowchart activity in the Designer panel.
- Insert an Input Dialog activity below Start node.
- Enter text "Vote Eligibility" in the *Title*, and "Enter your age:" in the *Label* text area.
- Create a variable **intUserAge** with Variable type as **Int32** and Scope as **Flowchart**. Assign this variable to an Input Dialog activity through the **Result** property in its **Properties** panel.
- Insert a **Flow Decision** activity below the Input Dialog activity. In its Properties panel, enter the condition **intUserAge >=18**
- Insert a Message box activity on the left node of the Flow Decision activity, and another Message box activity on the right node.
- In the Message box activity lying on the True branch of the Flow Decision, insert the text "You can vote!"
- In the Message box activity lying on the False branch of the Flow Decision, insert the text "You cannot vote!"
- Run the file and test few variations.

## Slide 48

## Loops in Flowcharts

Used when a certain part of the flowchart is required to perform a repetitive task based on a specified condition.




Loops are structures used to automate repetitive tasks. A certain section in the flowchart may be required to perform repetitive tasks and hence loops can be used inside the flowcharts. In flowcharts, the simplest types of loops can be created by connecting a certain point in the workflow to an earlier execution and repeating the task until the specified condition is met.

## Slide 49

### Classroom Exercise

---



Demonstrate how to create a **loop in Flowchart** by building a workflow that tells the user if the input is an even number or not.

- Ask the user to enter an even number.
- Continue asking for input until the number entered is an even number.
- If the number entered is an even number, then display in a message box "Perfect!"
- If the number entered is an odd number, then display in a message box "Try again."

Demonstrate how to create a loop in Flowchart. Ask the user to enter an even number, and until the right input is given, repeat the question.


- Insert a Flowchart activity in the Designer panel.
- Insert an Input Dialog activity below the Start node.
- Insert the text "Even Number" in the *Title*, and "Enter an even number:" in the *Label* text area.
- Create a variable **intEvenNo** with Variable type as **Int32** and Scope as **Flowchart**. Assign this variable to the Input Dialog activity through the **Result** property in its **Properties** panel.
- Insert a **Flow Decision** activity below the Input Dialog activity. In its Properties panel, enter condition **intEvenNo Mod 2 = 0**
- Insert a Message box activity on the left node of the Flow Decision activity, and another Message box activity on the right node.
- In the Message box activity lying on the True branch of the Flow Decision, insert the text "Perfect!"
- In the Message box activity lying on False branch of the Flow Decision, insert the text "Try Again!". Connect a node of this Message box activity with the Input Dialog activity to form a loop.

Run the file and test few variations.

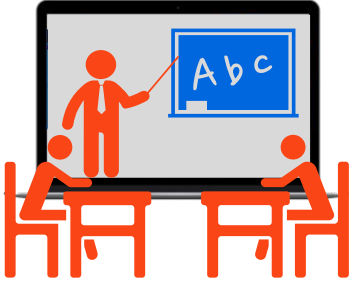


## Slide 50

## Practice Exercise



---



Build a workflow that asks user for his name and two-digit lottery number and displays if he is a winner.

- Ask name of the user and two-digit lottery number
- Give the user five chance to enter the correct lottery number
- If entry is below 54 and above 64, display "Enter your lottery number. X chance remaining." Here, X is the number of remaining chances for the user.
- If entry is between 54 and 64, display, "Congratulations User! You won the lottery." Replace User with the name of the user.
- If chances end before correct entry, display "Sorry, you lost. No more chances remaining"

Build a workflow that asks the user for their name and two-digit lottery number and displays if they are a winner.

- Ask the name of the user and two-digit lottery number
- Give the user five chance to enter the correct lottery number
- If the entry is below 54 and above 64, display "Enter your lottery number. X chance remaining." Here, X is the number of remaining chances for the user.
- If entry is between 54 and 64, display, "Congratulations User! You won the lottery." Replace the User with the name of the user.
- If chances end before correct entry, display "Sorry, you lost. No more chances remaining"

### Process Overview

- START
- Use an Input Dialog activity within a Flowchart activity to get the name of the user.
- Use another Input Dialog activity to take a lottery number from the user.
- Also, display remaining chances using the above Input Dialog activity in the format "Enter your lottery number. X chance remaining."
- Use a Flow Decision activity to check if the input is above 54.
  - If the input is below 54, use an Assign activity to increment the count of chances used by 1.
  - If the input is above 54, use another Flow Decision activity to check if the input is below 64.

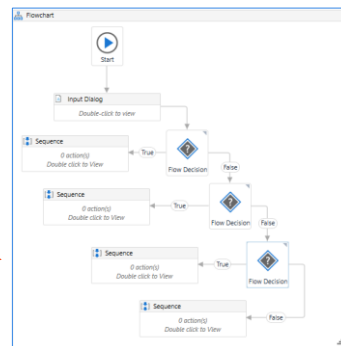
- If the input is above 64, use an Assign activity to increment the count of chances used by 1.
  - If the input is below 64, use a Message Box activity to display “Congratulations User! You won the lottery.” Replace User with the name of the user.
- Use a Flow Decision activity to check if the count of chances used by the user reaches five.
  - Use a Message Box activity to display, “Sorry, you lost. No more chances are remaining”.
- STOP

## Slide 51

## Nesting Flowcharts and Sequences

Nesting is used while creating complex workflows, as it allows a logical division of the program and promotes reusability.

Sequences nested  
within a flowchart  
activity



Nesting is used while creating complex workflows, as it allows a logical division of the program and promotes reusability. Although it is technically possible to nest flowcharts and sequences in every way, the only sustainable combination is to use a flowchart to represent the overall logic of the program and to use sequences for different parts inside.

## Slide 52

Sequences vs. Flowcharts	
The differences between sequences and flowcharts are:	
Sequences	Flowcharts
Suitable for linear processes	Suitable for complex processes
Seamless movement from one activity to another	Complex movement between two activities, but easier to understand as a whole
Acts as a single block of activity	Connects multiple block of activities through branching

The differences between sequences and flowcharts are:

**Sequences**

- Suitable for linear processes
- Seamless movement from one activity to another
- Acts as a single block of activity


**Flowcharts**


- Suitable for complex processes
- Complex movement between two activities, but easier to understand as a whole
- Connects multiple blocks of activities through branching

Slide 53

## Error Handling

- Errors
- Exceptions
- Error handling approach
- Error handling activities
  - Try Catch
  - Retry Scope
  - Global Exception Handler
- Continue on Error
- Best practices for error handling

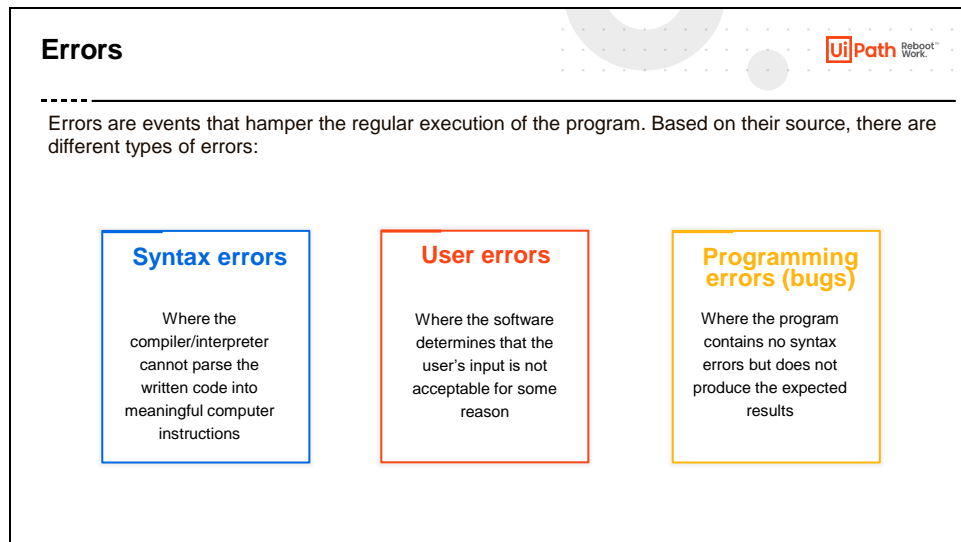




Reboot<sup>™</sup>  
Work.

This section introduces errors, exceptions and the best practices for handling the errors.

## Slide 54




An error is any type of event that prevents the regular execution of a program. Some of the errors completely stop the execution, others delay it, and others just interfere with them, without having a clear impact.

Based on their source, there are different types of errors:

- **Syntax errors**, where the compiler/interpreter cannot parse the written code into meaningful computer instructions.
- **User errors**, where the software determines that the user's input is not acceptable for some reason.
- **Programming errors**, where the program contains no syntax errors but does not produce the expected results. These are often called bugs.

## Slide 55



## Exceptions

-----

Exceptions are a subset of errors that are recognized (caught) by the program, categorized, and handled. The two types of exceptions are:

Application (System) Exception	Business Exception
<ul style="list-style-type: none"><li>• Describes an error rooted in a technical issue, such as an application that is not responding.</li><li>• Has a chance of being solved simply by retrying the transaction, as the application can unfreeze.</li><li>• Can be managed by following good naming conventions for activities and workflows. This helps in tracking the activity that caused the exception.</li></ul>	<ul style="list-style-type: none"><li>• Describes an error rooted in the fact that certain data which the automation project depends on is incomplete, missing, outside of set boundaries, or does not pass other data validation criteria.</li><li>• An exception from the usual process flow and the validation is made explicitly by the developer inside the workflow.</li><li>• The text in the exception should contain enough information for a human user (business user or developer) to understand what happened and what actions need to be taken.</li></ul>

Exceptions are a subset of errors that are serious enough to produce a material effect and for which there can be a mechanism to identify and address them. Exception handling mechanism refers to how to prevent and/or respond to exceptions. Sometimes, the handling mechanism can be simply stopping the execution. Some of the exceptions are linked to the systems used, while others are linked to the logic of the business process.

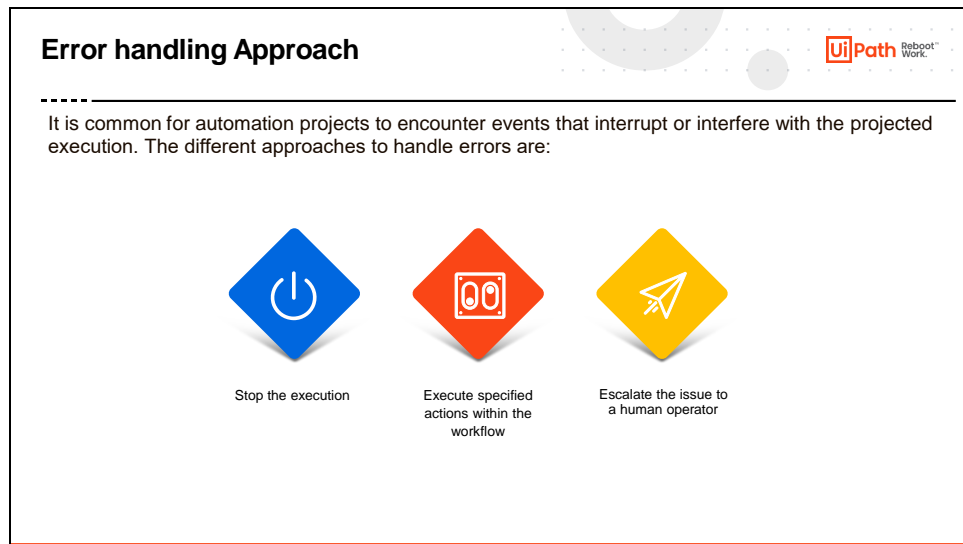
The types of exceptions are:

- **Application (System) Exception:**
  - Describes an error rooted in a technical issue, such as an application that is not responding.
  - Has a chance of being solved simply by retrying the transaction, as the application can unfreeze.
  - Can be managed by following good naming conventions for activities and workflows. This helps in tracking the activity that caused the exception.
- **Business Exception:**
  - Describes an error rooted in the fact that certain data which the automation project depends on is incomplete, missing, outside of set boundaries, or does not pass other data validation criteria.
  - It is an exception from the usual process flow and the validation is made explicitly by the developer inside the workflow.

- The text in the exception should contain enough information for a human user (business user or developer) to understand what happened and what actions need to be taken.



## Slide 56




It is common for automation projects to encounter events that interrupt or interfere with the projected execution. The different approaches to handle errors are:

- Stop the execution
- Execute specified actions within the workflow
- Escalate the issue to a human operator


## Slide 57

### Error handling Activities


UiPath Studio offers following three activities to handle errors. They are:



Try Catch



Retry Scope



Global Exception Handler

UiPath Studio offers following three activities to handle errors. They are:

- Try Catch
- Retry Scope
- Global Exception Handler

The subsequent slides gives more detail on these activities.

## Slide 58

## Try Catch

-----

Catches a specified exception type in a sequence or activity, and either displays an error notification or dismisses it and continues the execution.

**Try**

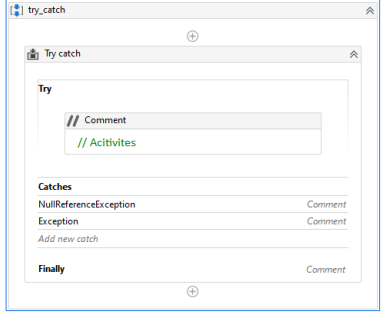
The activities performed which have a chance of throwing an error.

**Catches**

The activity or set of activities to be performed when an error occurs. Multiple errors and corresponding activities can be added in this block.

**Finally**

Holds an activity that should be executed only if no error occurred or if the error was already caught



The Try Catch activity is one of the most important activities used for exception handling purposes. It is generally used when specific parts of the project may trigger errors. The activity catches a specified exception type in a sequence or activity, and either displays an error notification or dismisses it and continues the execution.

As a mechanism, Try Catch runs the activities in the Try block and, if an error takes place, executes the activities in the Catch block. The Finally block is only executed when no exceptions are thrown or when an exception is caught and handled in the Catch block (without being re-thrown).


The activity has three main sections:

- Try - The activity performed which has a chance of throwing an error.
- Catches - The activity or set of activities to be performed when an error occurs.
  - Exception - The exception type to look for. Multiple exceptions can be added.
- Finally - Holds an activity that should be executed only if no error occurred or if the error was already caught.

## Slide 59

### Classroom Exercise

---



Demonstrate how to catch Error/Exception and continue running a code using a **Try Catch** activity. Create a workflow that:

- Inserts a text in an MS Word file in the first attempt.
- Encounters an error when attempting to write a second text.
- Catches and displays the error in a message box.
- Continues to save and close the MS Word file.

Demonstrate how to create a Robot that catches an Error/Exception and continues running using the **Try Catch** activity.

- Open MS Word application on the desktop called test.docx.
- Insert a **Type Into** activity in the designer panel.
- Click the “Indicate on screen” link of the **Type Into** activity and indicate the editor area of the MS Word file.
- In the text area of the Type Into activity, insert the text “Automation is the future”.
- Insert a **Try Catch** activity below the first **Type Into** activity.
- Insert a **Type Into** activity within the **Try** section of the **Try Catch** activity.
- Click the “Indicate on screen” link of the **Type Into** activity and indicate the editor area of the MS Word file.
- In the text area of the **Type Into** activity, insert “[k(enter)] World is changing”.
- Intentionally break the selector of this activity. Click the hamburger button and select **Edit Selector** from the context menu. Change title to test2.docx, click OK. This will create an error.

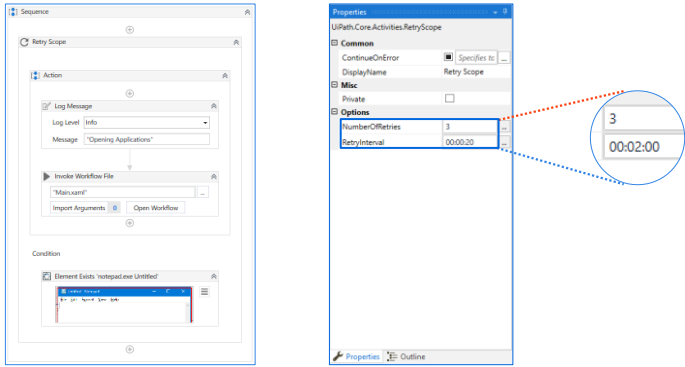
- To handle this exception, add **System.Exception** from drop down in the **Choose** section of the **Try Catch** activity.
- Insert a **Message Box** within the **Choose** section. Insert the text **exception.Message** in the text area.
- In the **Finally** section, insert two **Click** activities.
- Indicate the **Save** button of the MS Word file with the first click activity, and the **Close** button with the second click activity.
- Let's run the program...
  - First text gets written in MS Word.
  - Process trying to write the second text...but the error message box shows up...click **OK**.
  - Process now clicks the Save button and the Close button of the MS Word and closes the application.
  - Open the closed MS Word file to check the written text.

**Outcome:** We can see that the first text gets written, but the second text gives an error. However, still the program runs further and closes the MS Word application. It means our program did not stop even after facing the error.

## Slide 60

## Retry Scope

Retries the contained activities as long as the condition is not met, or an error is thrown.



Retry Scope activity retries the contained activities as long as the condition is not met, or an error is thrown.

It is used to retry the execution in situations in which an error is expected. The execution will be retried until a certain event happens (for a number of times) or without any condition (retried until no exception is thrown).

It is used for catching and handling an error, which is why it's similar to Try Catch. The difference is that this activity simply retries the execution instead of providing a more complex handling mechanism.

Some of the properties of Retry Scope are:

- **NumberOfRetries** - The number of times that the sequence is to be retried.
- **RetryInterval** - Specifies the amount of time (in seconds) between each retry.

For example, consider a website that simply works faulty, and the user just needs to click the same button over and over until it goes to the desired screen.

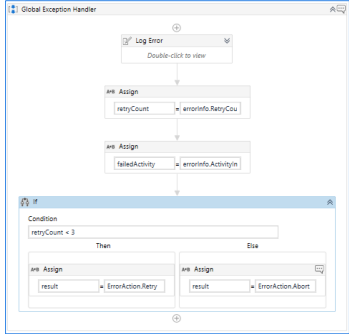
To know more, visit: <https://docs.uipath.com/activities/docs/retry-scope>

## Slide 61

## Global Exception Handler

A type of workflow designed to determine the behavior when encountering an execution error at the project level.

- Only one Global Exception Handler can be set per automation project.
- It is used in conjunction with Try Catch and only uncaught exceptions will reach the Exception Handler.
- It is not available for library projects, only processes.



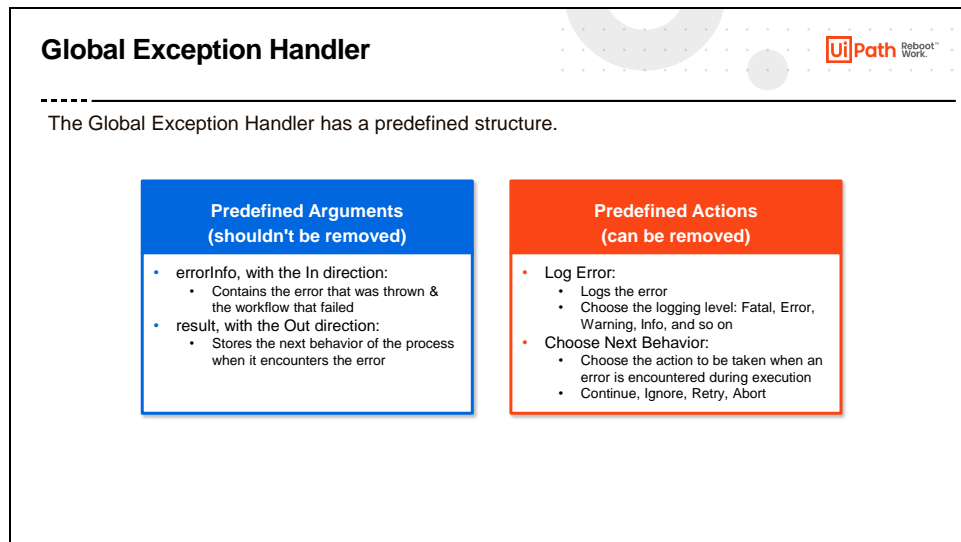
The Global Exception Handler is a type of workflow designed to determine the behavior when encountering an execution error at the project level. This is why only one Global Exception Handler can be set per automation project.

The Global Exception Handler is used in conjunction with Try Catch. Only uncaught exceptions will reach the Exception Handler. If an exception occurs inside a Try Catch activity and it is successfully caught and treated inside the Catch block (and not re-thrown), it will not reach the Global Exception Handler.

It is not available for library projects, only processes.

A Global Exception Handler can be created either by starting a new project with this type, or by setting an existing project as Global Exception Handler from the Project panel.

## Slide 62



The Global Exception Handler has a predefined structure.

It has two **predefined arguments** (that shouldn't be removed):

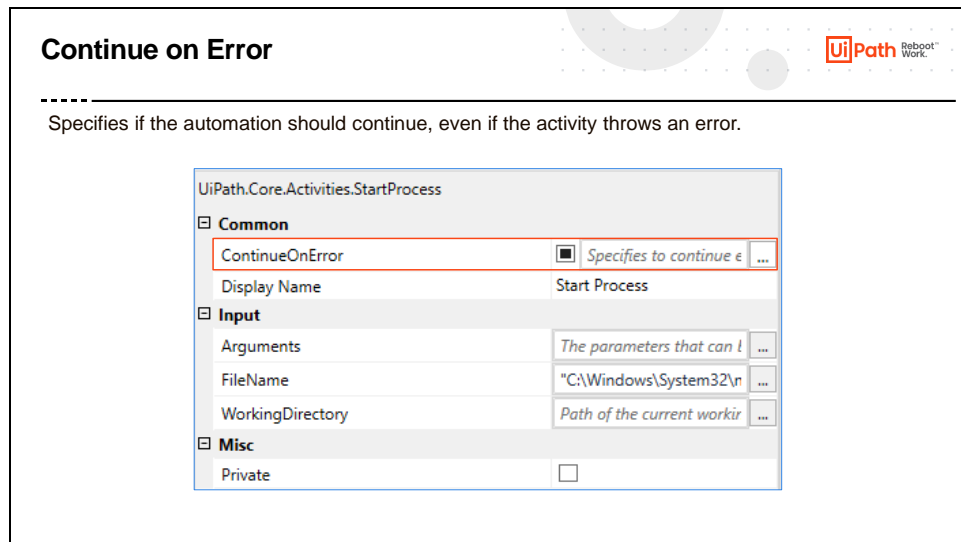
- **errorInfo**, with the In direction - contains the error that was thrown and the workflow that failed
- **result**, with the Out direction - will store the next behavior of the process when it encounters the error

It contains two **predefined actions** (that can be removed, and other actions can be added):

- **Log Error:** This part simply logs the error. The developer gets to choose the logging level: Fatal, Error, Warning, Info, and so on.
- **Choose Next Behavior:** Here the developer can choose the action to be taken when an error is encountered during execution:
  - **Continue** - The exception is re-thrown
  - **Ignore** - The exception is ignored, and the execution continues from the next activity
  - **Retry** - The activity which threw the exception is retried
  - **Abort** - The execution stops after running the current handler



## Slide 63



Continue On Error is very useful for activities that work with UI interactions. It is the easiest way to tell the workflow to continue even if the activity throws an error.

This field only supports Boolean values (True, False). The default value in this field is False. As a result, if this field is blank and an error is thrown, the execution of the project stops. If the value is set to True, the execution of the project continues regardless of any error.

When it is set to True on an activity that has a scope (such as Attach Window or Attach Browser), then all the errors that occur in other activities inside that scope are also ignored.

It is used when:

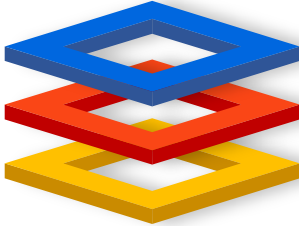
- Using data scraping, so that the activity doesn't throw an error on the last page (when the selector of the 'Next' button is no longer found);
- The user is not interested in capturing the error, but simply in the execution of the activity.




## Slide 64

### Best Practices for Error Handling

-----

The best practices for handling errors are:



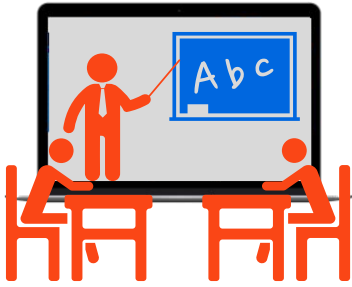
-  Breaking the process into smaller workflows
-  Using Try/Catch blocks
-  Using Global Exception Handler

The best practices for handling errors are:

- Breaking the process into smaller workflows for a better understanding of the code, independent testing and reusability.
- Using Try/Catch blocks to predict and handle exceptions.
- Using Global Exception Handler for situations that are global and/or less probable to happen.

## Slide 65

### Practice Exercise



- Build a workflow using a **Try Catch** activity to do the following:
  - Take the Name, Gender, and Age as the user input.
  - Subtract current year with Age value to get the Year of Birth.
  - Handle an error that occurs due to a reckless user input of an incorrect age containing the 11-digit number.
  - Continue the process to display the Name, Gender, and Year of Birth of the user in a message box.

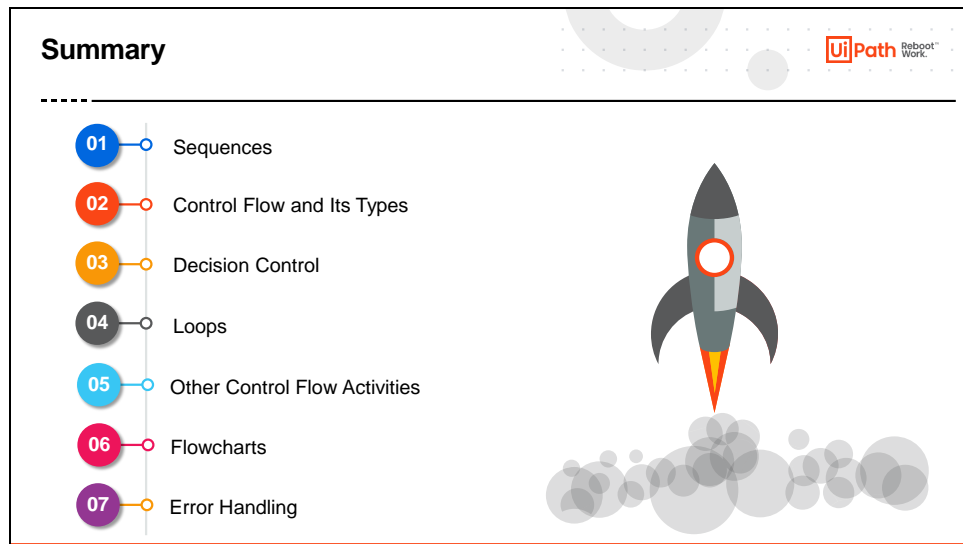
Build a workflow using a **Try Catch** activity to perform the following:

- Take the Name, Gender and Age as the user input.
- Subtract current year with Age value to get the Year of Birth.
- Handle an error that occurs due to a reckless user input of an incorrect age containing the 11-digit number.
- Continue the process to display the Name, Gender, and Year of Birth of the user in a message box.

### Process Overview

- START
- Use three Input Dialog activities within the Try Catch activity to ask for the Name, Gender, and Age of the user.
- Use an Assign activity to subtract the age from the current year to get the year of birth of the user
- Use Exception Type: System.Exception in the Catches section of the Try Catch activity to handle reckless input from the user. Store error in a string variable.
- Use a Message Box activity to display the Name, Gender, and Year of Birth of the user along with the Error, if any.
- STOP

## Slide 66



To summarize, this lesson explained:

- Sequences
- Control Flow and Its Types
- Decision Control
- Loops
- Other Control Flow Activities
- Flowcharts
- Error Handling