GUVI

tech deserves you

RPA Design & Development V2.0

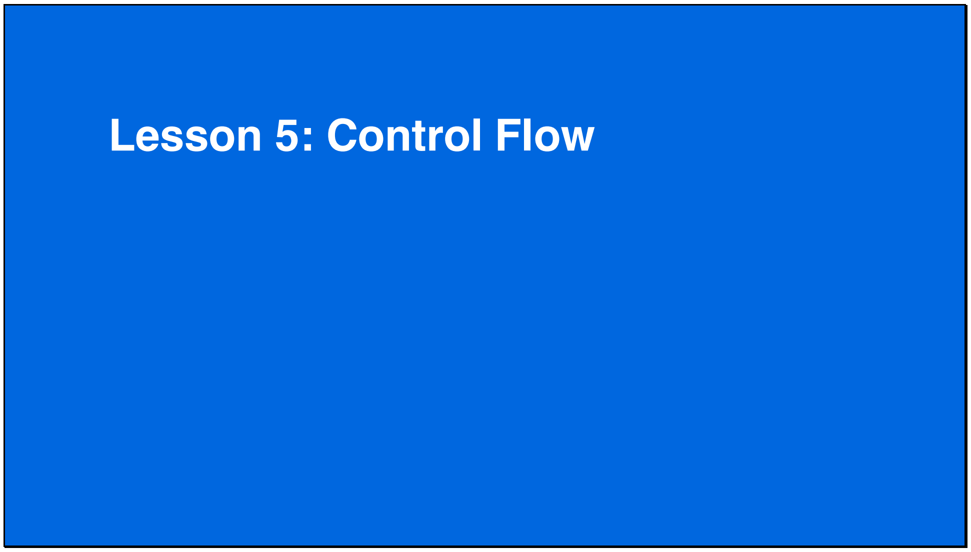
Student Manual

UiPath™ Learning Partner

RPA Design and Development V2.0

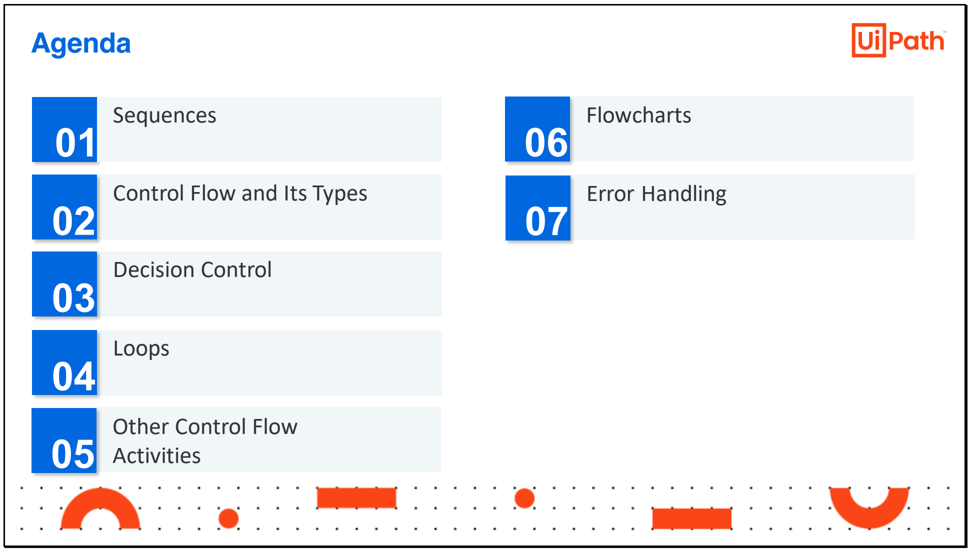
UiPath

Welcome to ‘RPA Design and Development Course’.



Lesson 5: Control Flow

The fifth lesson of this course is Control Flow.



Agenda

UiPath

01 Sequences

02 Control Flow and Its Types

03 Decision Control

04 Loops

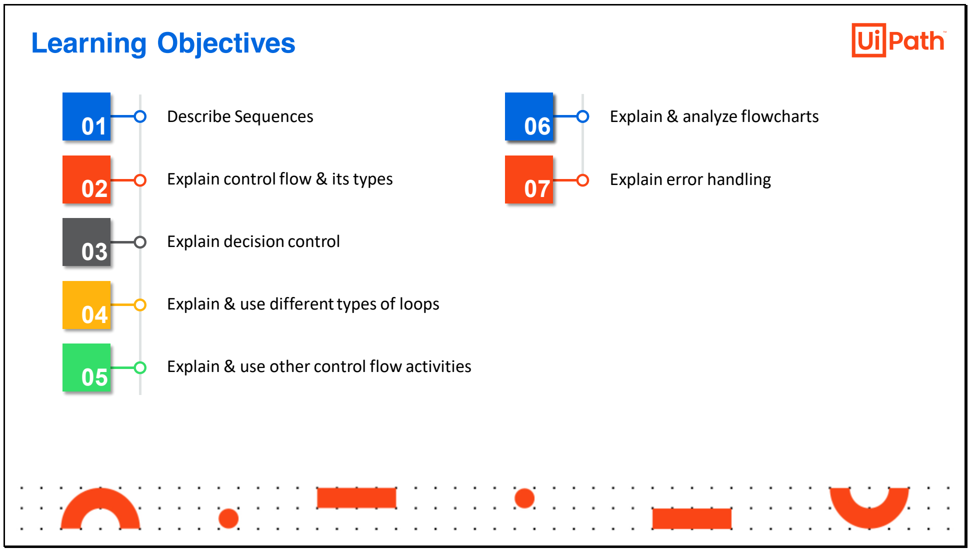
05 Other Control Flow Activities

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



Learning Objectives

UiPath

01 Describe Sequences

02 Explain control flow & its types

03 Explain decision control

04 Explain & use different types of loops

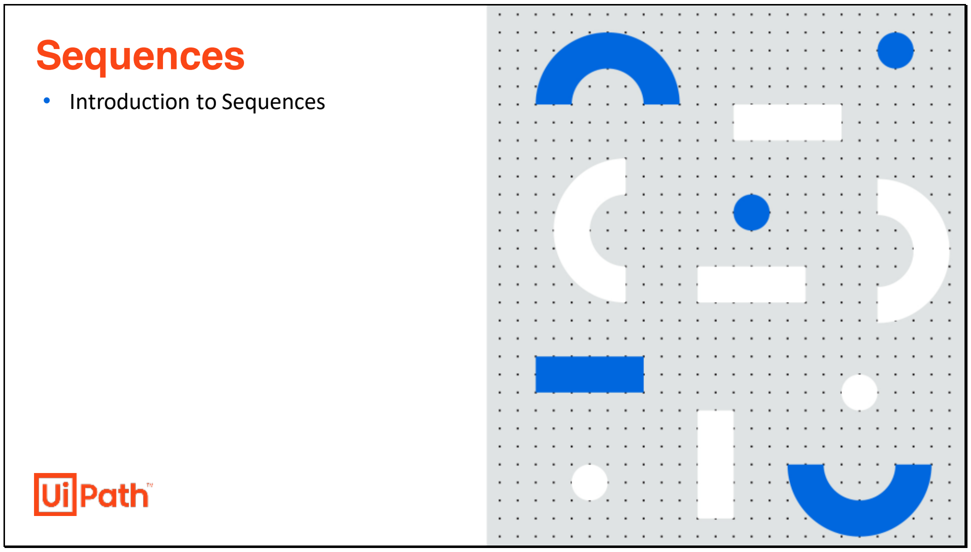
05 Explain & use other control flow activities

06 Explain & analyze flowcharts

07 Explain error handling

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

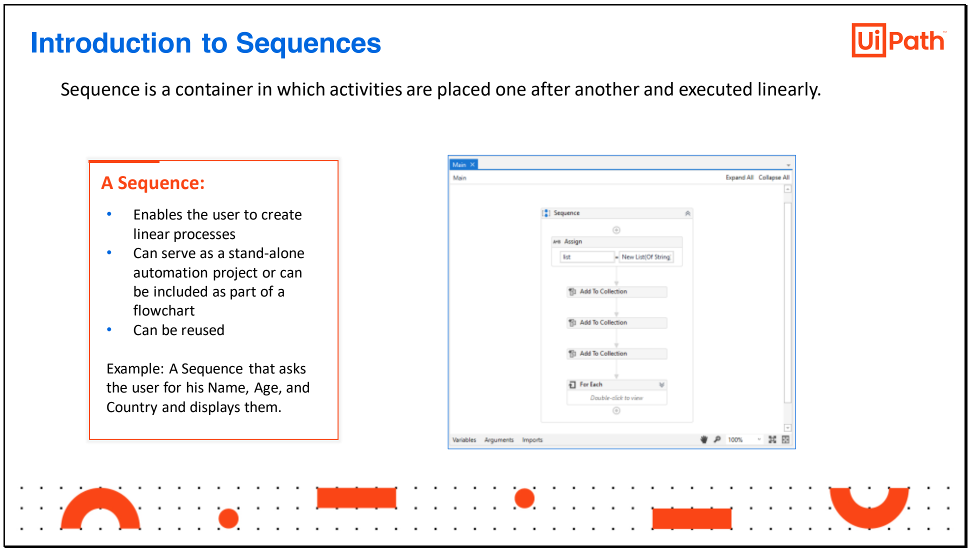
- Describe Sequences
- Explain control flow & its types
- Explain decision control
- Explain & use different types of loops
- Explain & use other control flow activities
- Explain & analyze flowcharts
- Explain error handling



Sequences

- Introduction to Sequences

This section gives an overview of Sequences.



Introduction to Sequences

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

A Sequence:

- Enables the user to create linear processes
- Can serve as a stand-alone automation project or can be included as part of a flowchart
- Can be reused

Example: A Sequence that asks the user for his Name, Age, and Country and displays them.

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.

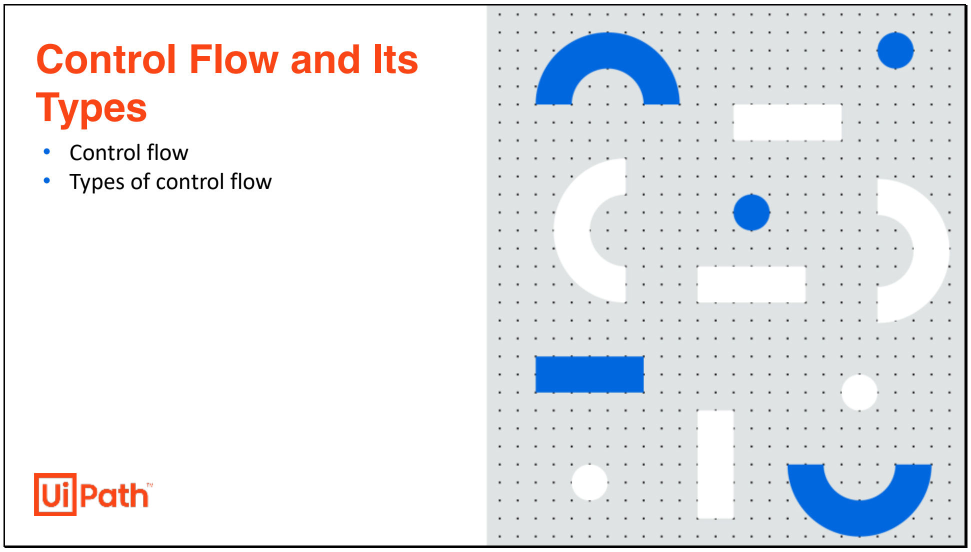
One of the key features of Sequences is that they can be reused time and again. A Sequence can either serve as a stand-alone automation project or can be included as part of a flowchart or a state machine to help group specific activities.

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

- Name
- Age
- Country

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

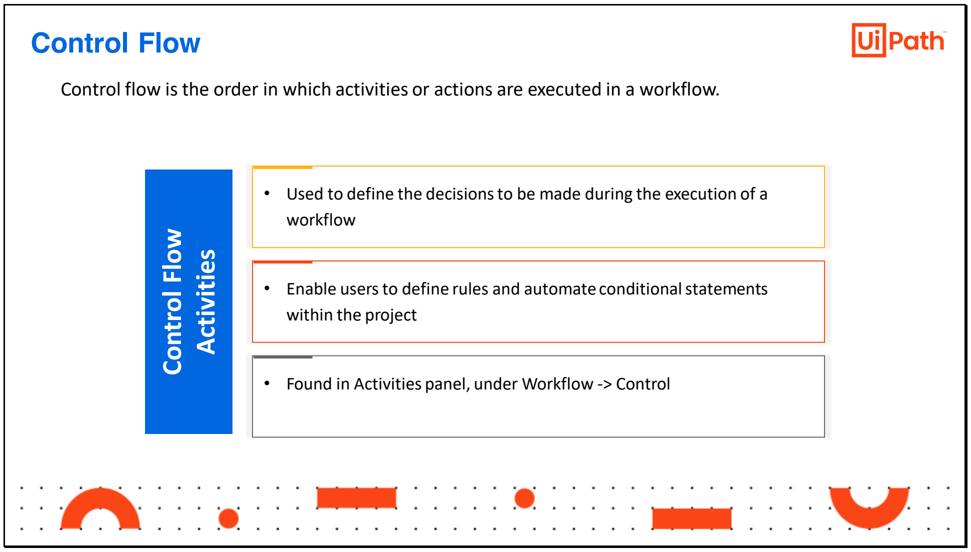
Refer: https://docs.uipath.com/studio/docs/sequences for additional detail on this example.



Control Flow and Its Types

- Control flow
- Types of control flow

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



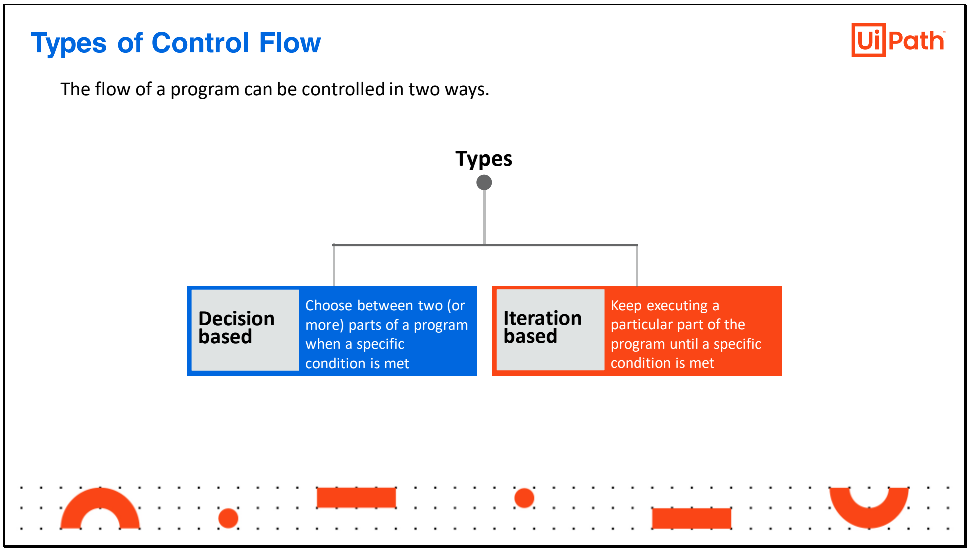
Control Flow

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 Activities panel, under Workflow -> Control

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 help the user to choose a suitable path to follow in the workflow. These activities are found in Activities panel, under Workflow -> Control.



Types of Control Flow

The flow of a program can be controlled in two ways.

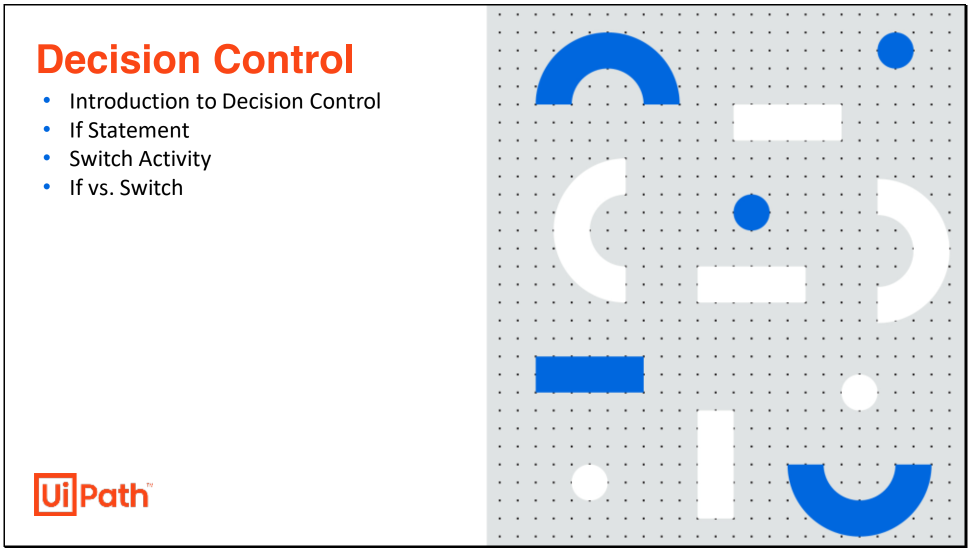
UiPath
Types
Decision based
Choose between two (or more) parts of a program when a specific condition is met
Iteration based
Keep executing a particular part of the program until a specific condition is met

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.

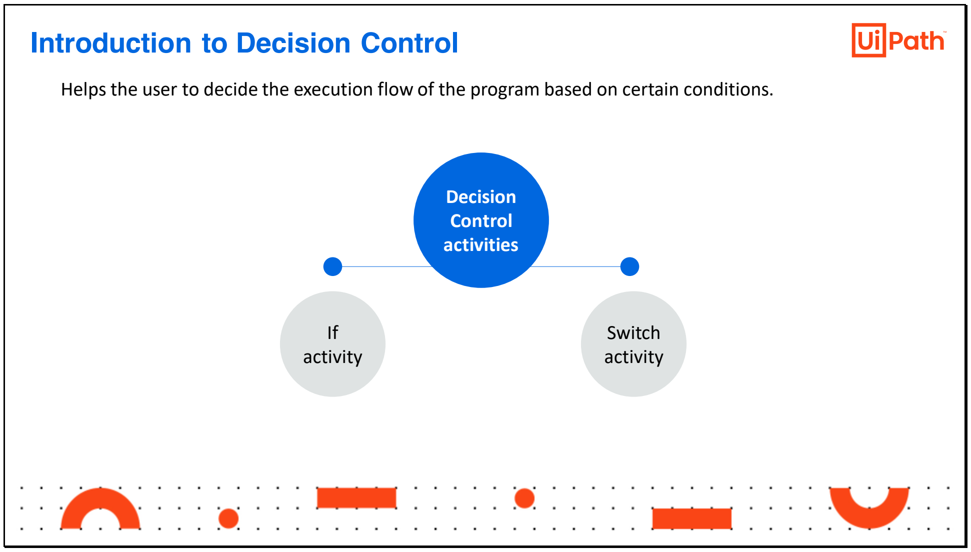
These activities are discussed in detail in the coming sections.



Decision Control

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

This section explains decision control in detail.



Introduction to Decision Control

Helps the user to decide the execution flow of the program based on certain conditions.

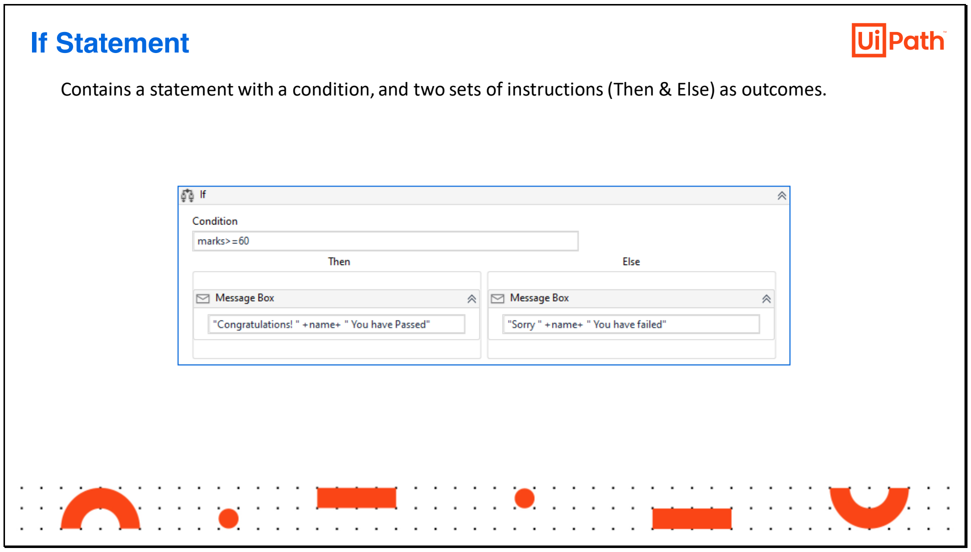
Decision Control activities

If activity

Switch activity

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**.



If Statement

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

Condition: marks>=60

Then – Message Box: "Congratulations! " +name+ " You have Passed"

Else – Message Box: "Sorry " +name+ " You have failed"

The If statement 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

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

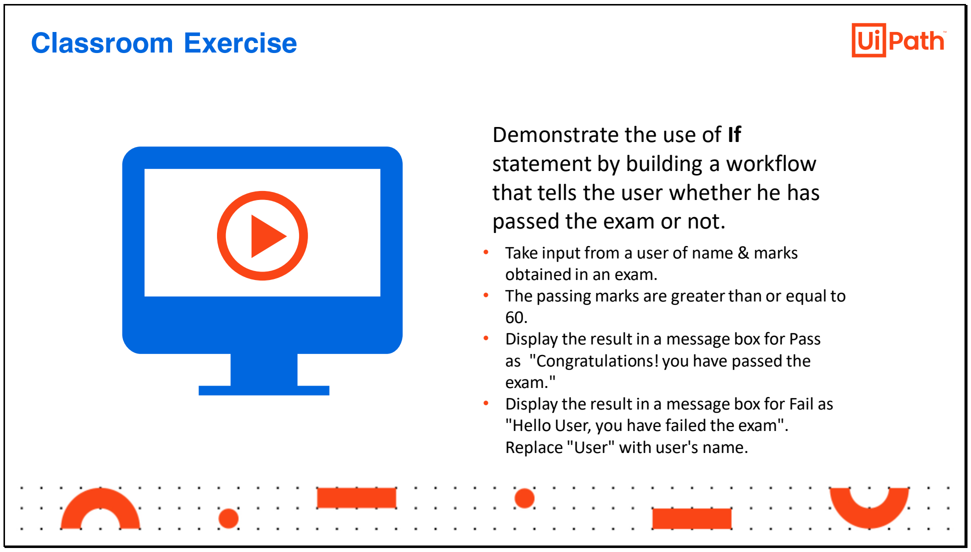
Example:
An automation to input a candidate's name & marks obtained in an exam and decide if he has passed the exam or not. Here, the condition is specified in If activity which displays the result accordingly:

• if the marks are >= 6o, then

Congratulations! 'name' You have passed

• Else

Sorry, 'name' You have failed



Classroom Exercise

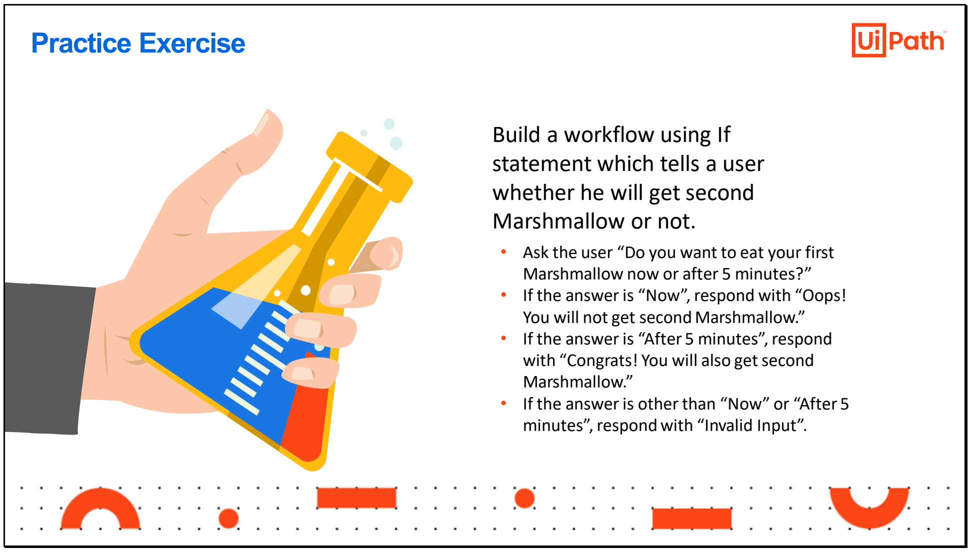
Demonstrate the use of **If** statement by building a workflow that tells the user whether he has passed the exam or not.

- Take input from a user of name & marks obtained in an exam.
- 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 user's name.

Demonstrate the use of If statement by building a workflow that tells the user whether he 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 **names** 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 **marks** for this activity with Variable Type as **Int32** using **Variables** panel.
- Enter this variable in **Result** property of this **Input Dialog** activity using its **Properties** panel.
- Insert **If** activity below the second Input Dialog activity.
- Enter condition **marks >= 60**.
- Insert two **Message Box** activities. One in **Then** box and another in **Else** box.
- In the first Message Box activity, enter text **“Congratulations!” + names + “, you have passed the exam.”**
- In the second Message activity, enter text **“Hello”** + **names** + **“, you have failed the exam.”**
- Run the process
- Enter 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.



Practice Exercise

UiPath

Build a workflow using If statement which tells a user whether he will get second Marshmallow or not.

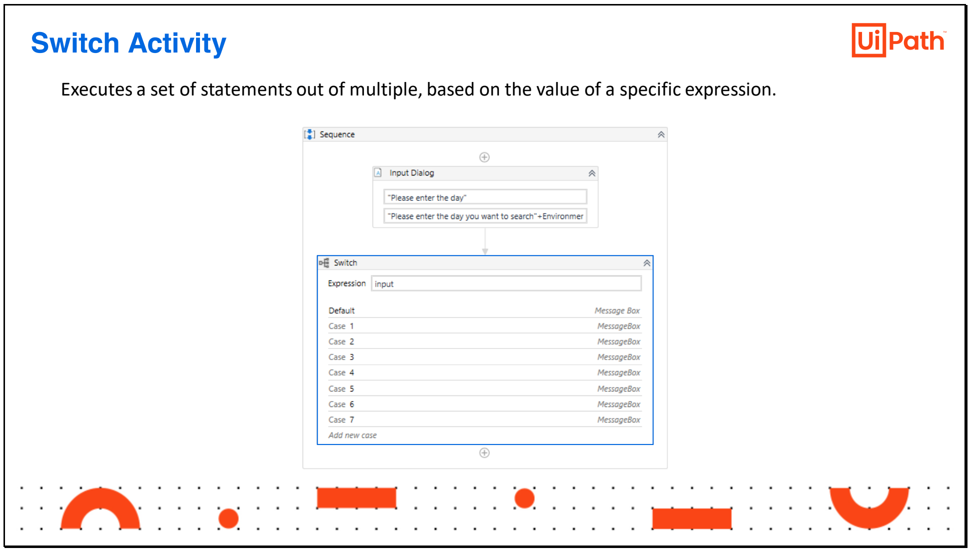
- Ask the user “Do you want to eat your first Marshmallow now or after 5 minutes?”
- If the answer is “Now”, respond with “Oops! You will not get second Marshmallow.”
- If the answer is “After 5 minutes”, respond with “Congrats! You will also get second Marshmallow.”
- If the answer is other than “Now”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

Build a workflow using If statement which tells a user whether he will get second Marshmallow or not.

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

Algorithm

- START
- Use an Input Dialog activity to ask 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 second Marshmallow.”
    - If the answer is “After 5 minutes”, use a Message Box activity to display “Congrats! You will also get second Marshmallow.”
    - If the answer is other than “Now” or “After 5 minutes”, use a Message Box activity to display “Invalid Input”.
- STOP



Switch Activity

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

"Please enter the day"

"Please enter the day you want to search"+Environmer

Expression: input

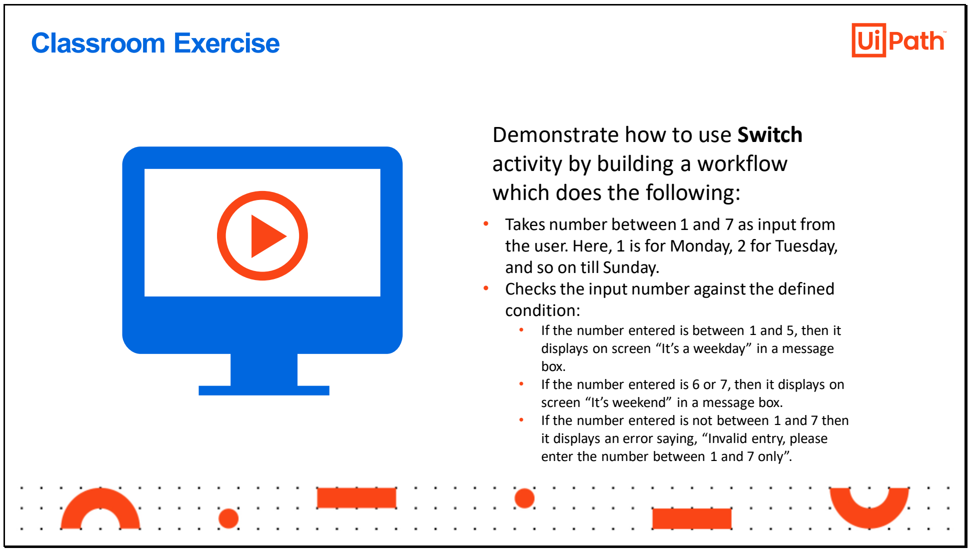
Default	Message Box
Case 1	MessageBox
Case 2	MessageBox
Case 3	MessageBox
Case 4	MessageBox
Case 5	MessageBox
Case 6	MessageBox
Case 7	MessageBox
Add new case	

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 actions 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 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.

Switch activity can be used it 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.



Classroom Exercise

UiPath

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

- Takes number 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 screen "It's a weekday" in a message box.
  - If the number entered is 6 or 7, then it displays on 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".

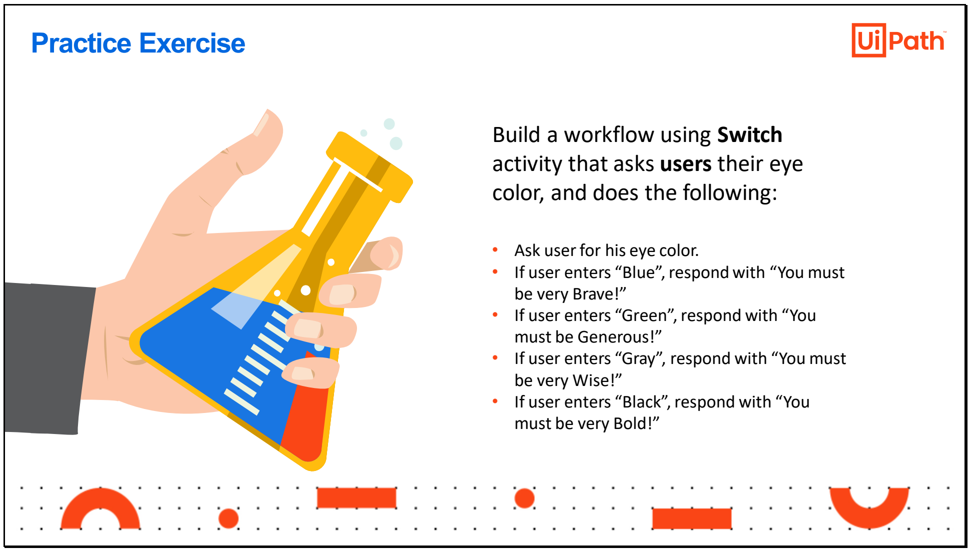
Demonstrate how to use Switch activity in workflows.

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

- Insert **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 **dayNumber** for the Input Dialog activity using Variables panel with **Variable Type** as **Int32**.
- Enter this variable in **Result** property of this activity using its **Properties** panel.
- Insert a **Switch** activity below Input Dialog activity.
- Insert the variable **dayNumber** for *Expression*.
- Insert a **Message Box** activity in *Default* section of Switch activity and enter 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 text "It's a weekday".
- Repeat same process for 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.



Practice Exercise

UiPath

Build a workflow using **Switch** activity that asks **users** their eye color, and does the following:

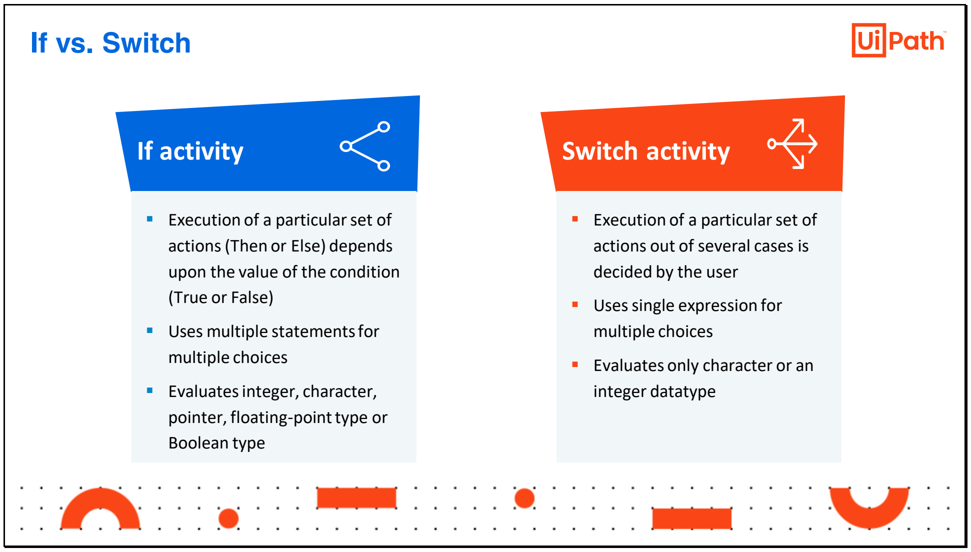
- Ask user for his eye color.
- If user enters "Blue", respond with "You must be very Brave!"
- If user enters "Green", respond with "You must be Generous!"
- If user enters "Gray", respond with "You must be very Wise!"
- If user enters "Black", respond with "You must be very Bold!"

Build a workflow **Switch activity** that asks users their eye color, and does the following:

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

Algorithm

- START
- Use Input Method activity to get the eye color input of the user.
- Use Switch activity to compare input with four different cases – Blue, Green, Gray, and other values.
- 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



If vs. Switch

UiPath

If activity

- Execution of a particular set of actions (Then or Else) depends upon the value of the condition (True or False)
- Uses multiple statements for multiple choices
- Evaluates integer, character, pointer, floating-point type or Boolean type

Switch activity

- Execution of a particular set of actions out of several cases is decided by the user
- Uses single expression for multiple choices
- Evaluates only character or an integer datatype

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

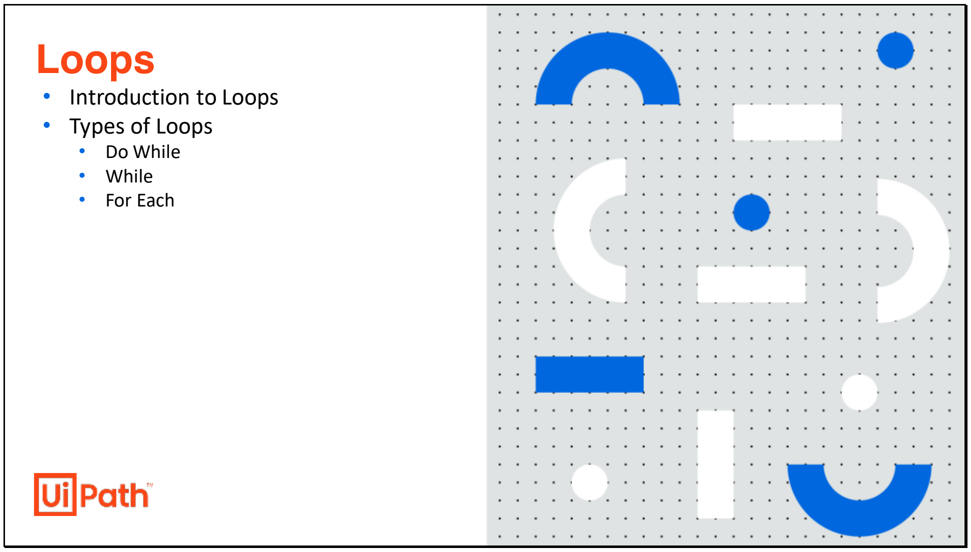
If activity:

- The execution of a particular set of actions (Then or Else) depends upon the value of the condition (True or False)
- Uses multiple statements for multiple choices
- Evaluates integer, character, pointer, floating-point type or Boolean type

Switch activity:

- The execution of a particular set of actions is decided by the user
- Uses single expression for multiple choices
- Evaluates only character or an integer datatype

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

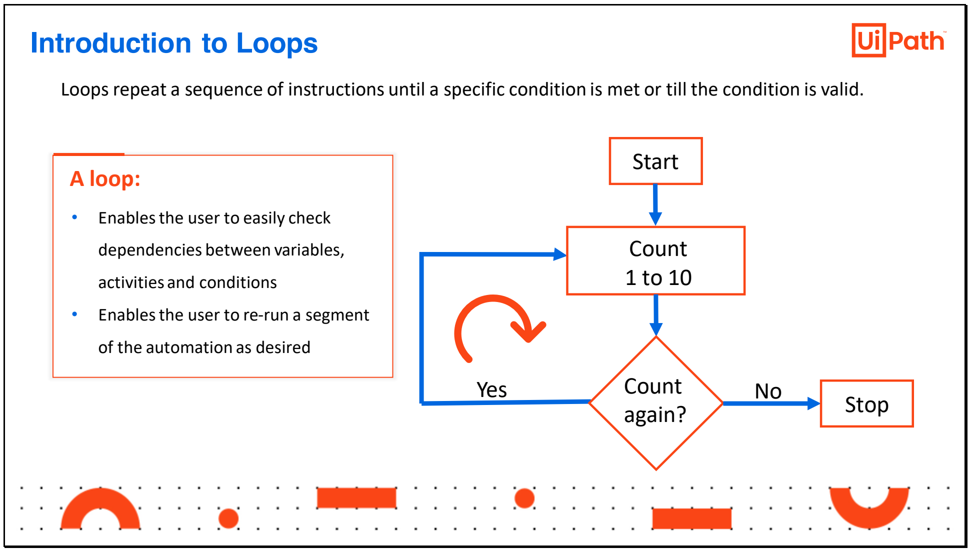


Loops

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

UiPath

This section explains loops and its several types in detail.



Introduction to Loops

UiPath

Loops repeat a sequence of instructions until a specific condition is met or till the condition is valid.

A loop:

- Enables the user to easily check dependencies between variables, activities and conditions
- Enables the user to re-run a segment of the automation as desired

Start

Count 1 to 10

Count again?

Yes

No

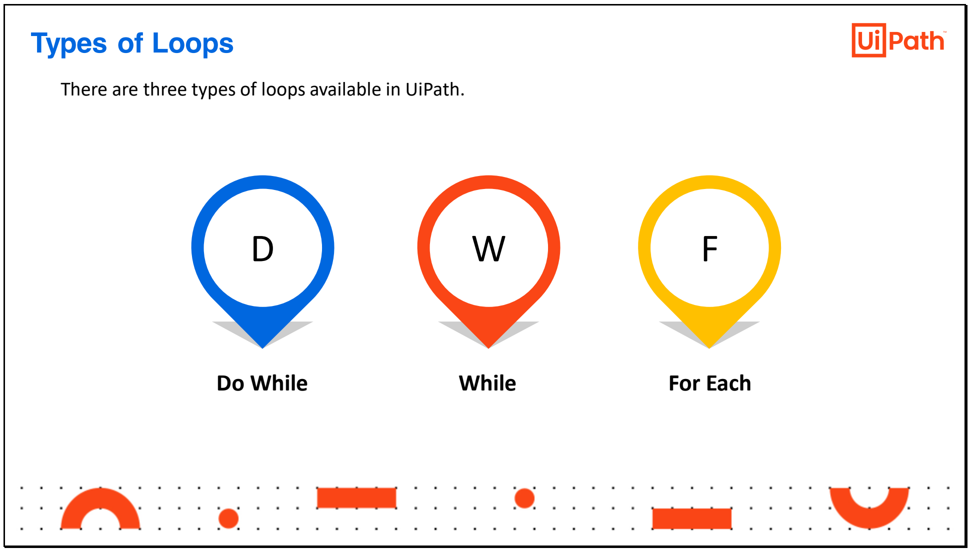
Stop

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 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.

Loops represent an important part of automations as they enable the user to easily check dependencies between variables, activities and conditions. Once created, they enable the user to run a segment of the automation for a specific number of times, until a condition is met, or indefinitely.



Types of Loops

UiPath

There are three types of loops available in UiPath.

Do While

While

For Each

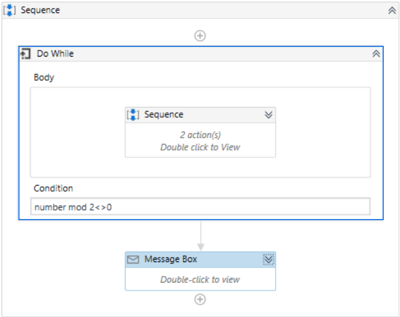
There are three types of loops in UiPath:

- **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.

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.



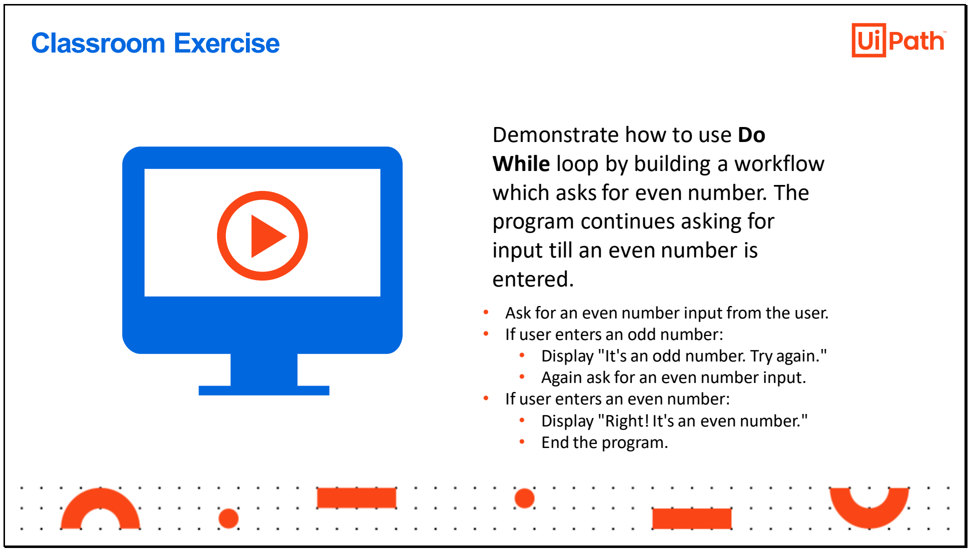
number mod 2<>0

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).



Classroom Exercise

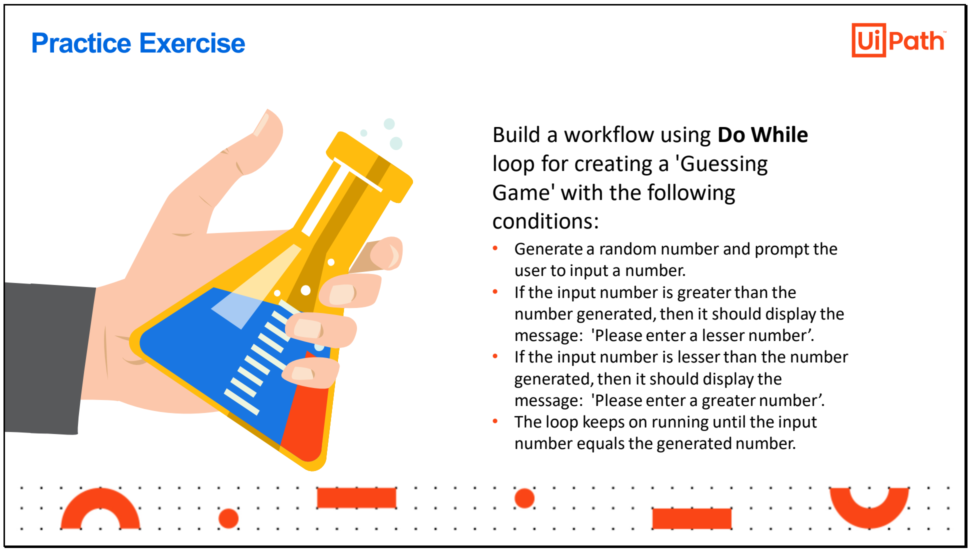
Demonstrate how to use **Do While** loop by building a workflow which asks for even number. The program continues asking for input till an even number is entered.

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

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

- Insert **Do While** activity in the designer panel.
- **Do While** activity will run the loop and execute the statement depending on the specified condition.
- Insert **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 **number** for this activity using **Variables** panel. Set Variable Type as **Int32**.
- Enter this variable in **Result** property of this activity using its **Properties** panel.
- Insert **If** activity below Input Dialog activity. In Condition box enter **number mod 2=0**.
- In Then section, insert a Message Box activity. Enter text "Right! It's an even number."
- In Else section, insert another Message Box activity. Enter text "It's an odd number. Try again."
- In the condition box of While activity, enter **number 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.



Practice Exercise

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

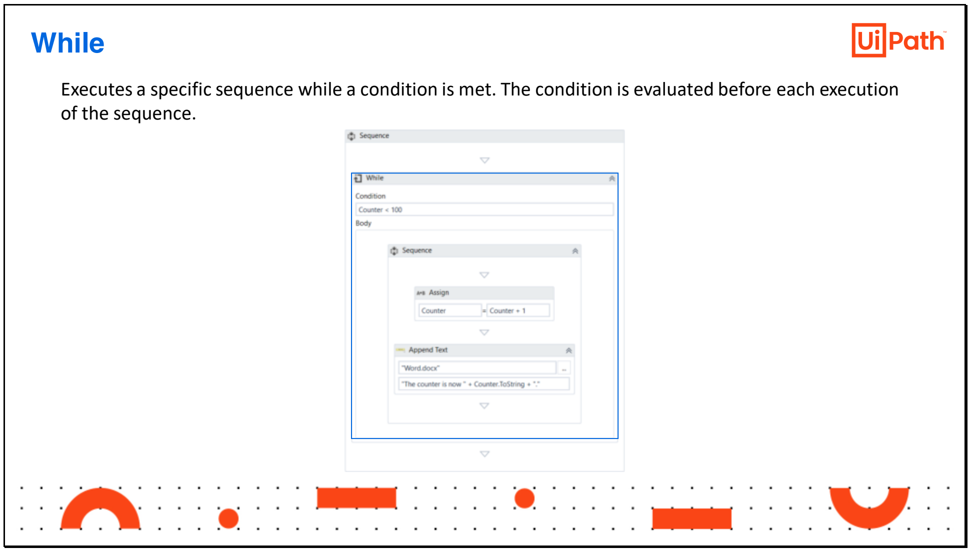
- Generate a random number and prompt the user to input a number.
- If the input number is greater than the number generated, then it should display the message: 'Please enter a lesser number'.
- If the input number is lesser than the number generated, then it should display the message: 'Please enter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

Build a workflow using **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 the generated number.

Algorithm

- START
- Declare the variables as 'RandomNo' , 'GuessedNo'
- Add decision box to the **Flowchart** and put the condition as GuessedNo < RandomNo add the message box to the true block and make the loop to the input box
- Add other decision box to the false branch and put the condition as GuessedNo = RandomNo
- In the true branch of the decision box add "Congratulations you Guessed Correct No."
- In the false branch of the decision box add '"You guessed a higher No. Guess again!!"'
- STOP



While

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

Counter < 100

Counter = Counter + 1

"Word.docx"

"The counter is now " + Counter.ToString + "."

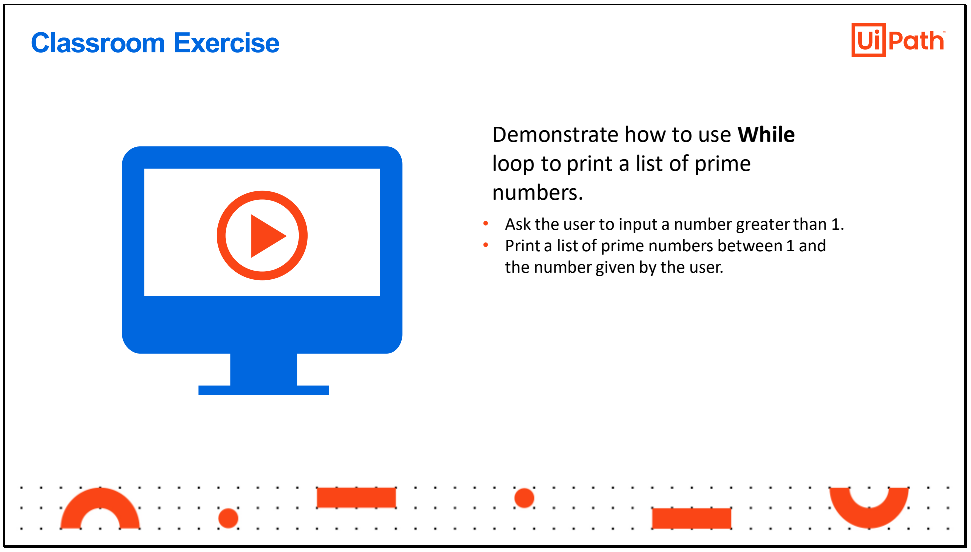
The While activity enables the user to create a loop that executes a specific process repeatedly, while a specific condition is met. The condition is evaluated before each execution of the sequence.

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 upto a given number.

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



Classroom Exercise

UiPath

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

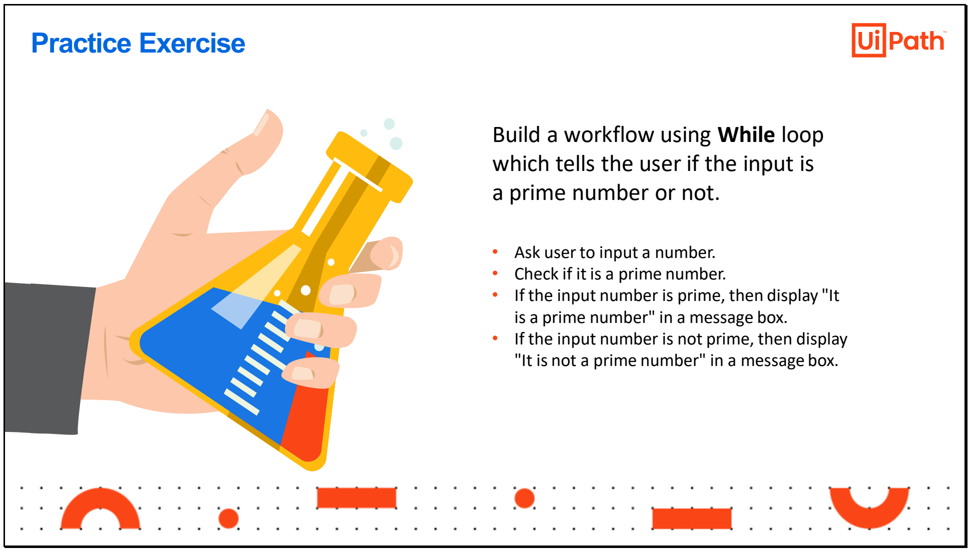
- Ask the user to input a number greater than 1.
- Print a list of prime numbers between 1 and the number given by the user.

Demonstrate how to use **While** loop.

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

- Insert **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 **number** for this activity using **Variables** panel. Set Variable Type as **Int32**.
- Enter this variable in **Result** property of this activity using its **Properties** panel.
- Insert **Assign** activity below the Input Dialog activity.
- In *To* text box, press **Ctrl+K** and enter **i** as a variable, and **1** in the adjacent box. In Variables panel, set Variable Type as **Int32** for **i**.
- Insert **While** activity below Assign activity.
- In the **Condition** box of While activity enter **i<=number**.
- In the **Body** section of While activity, insert **Write Line** activity.
- Enter the expression **i.ToString** in the text box.
- Insert **Assign** activity below **Write Line** activity.
- In *To* text box enter the variable **i**, and enter **i+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.



Practice Exercise

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

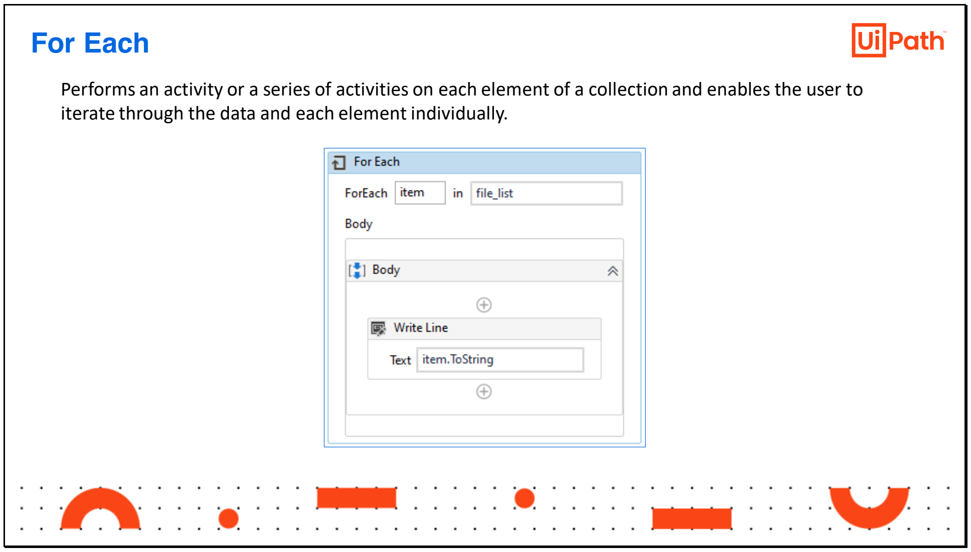
- Ask user to input a number.
- Check if it is a prime number.
- If the input number is prime, then display "It is a prime number" in a message box.
- If the input number is not prime, then display "It is not a prime number" in a message box.

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

- Ask 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.

Algorithm

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



For Each

Performs an activity or a series of activities on each element of a collection and enables the user to iterate through the data and each element individually.

For Each

ForEach item in file_list

Body

Body

Write Line

Text item.ToString

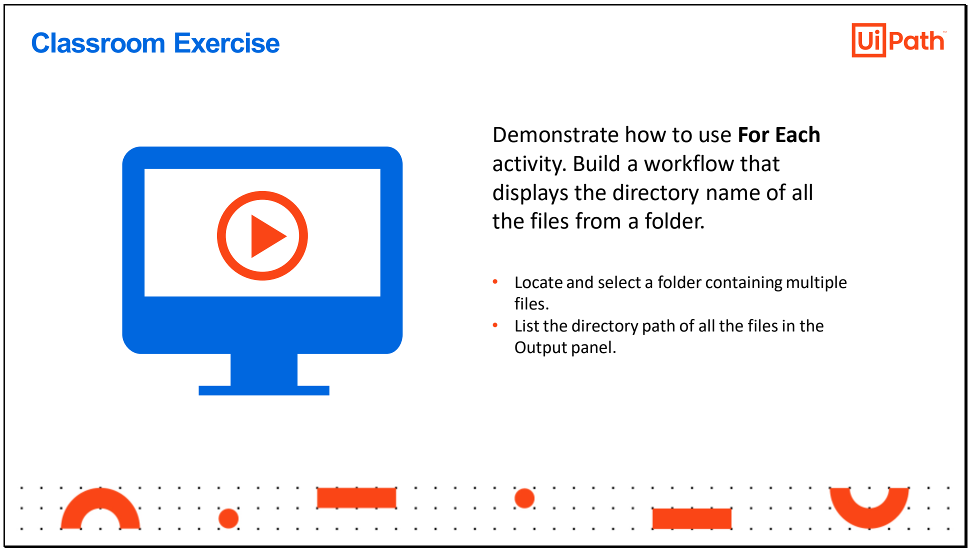
It performs an activity or a series of activities on each element of a collection.

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 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.

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



Classroom Exercise

UiPath

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

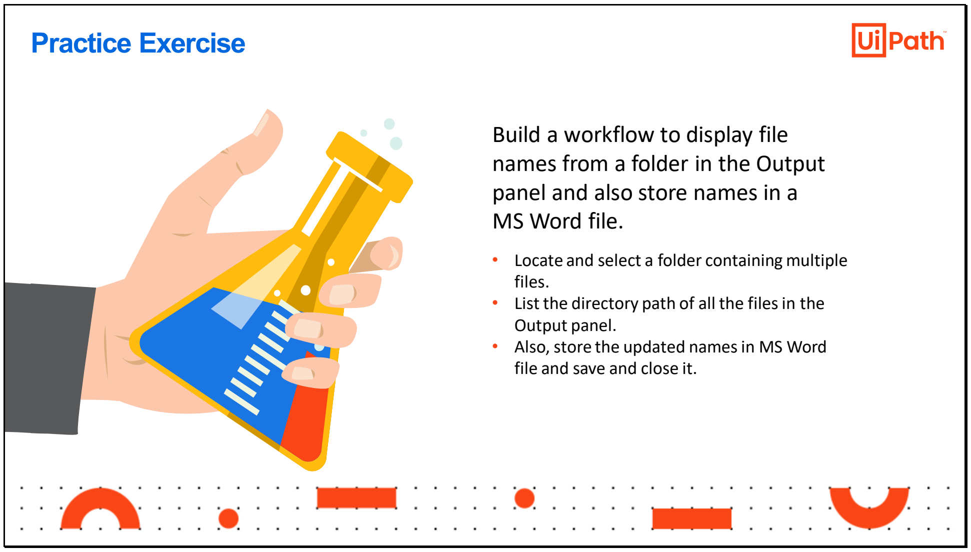
- Locate and select a folder containing multiple files.
- List the directory path of all the files in the Output panel.

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

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

- Insert **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 **Assign** activity below **Select Folder** activity.
- In *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 **For each** activity below **Assign** activity.
- Insert the text **item** in the first text box and the variable **fileList** in the second text box.
- Insert **Write Line** activity in the body section of **For Each** activity.
- Enter the text **item.ToString** in it.
- Run the file.
- From the window select a folder containing some files.
- Go to Output panel to see result

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



Practice Exercise

Build a workflow to display file names from a folder in the Output panel and also store names in a MS Word file.

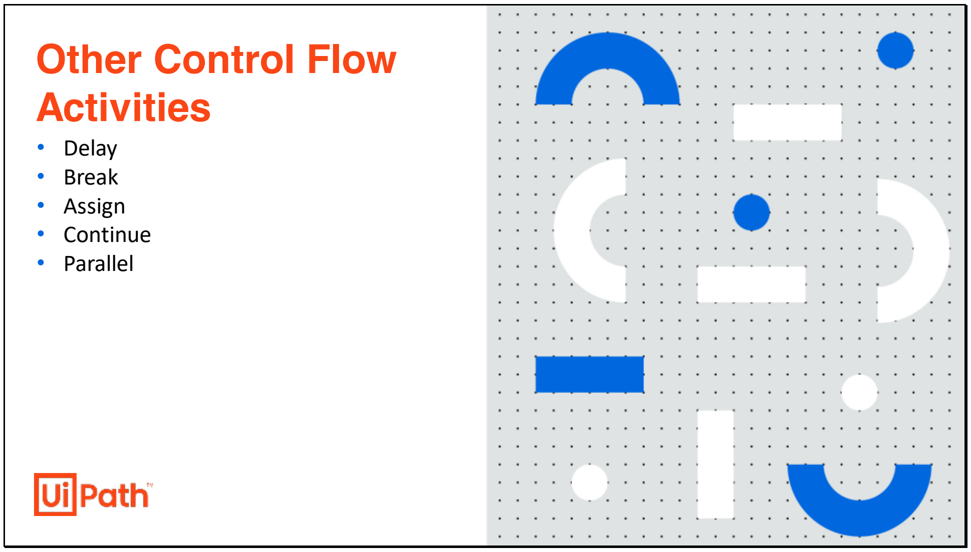
- Locate and select a folder containing multiple files.
- List the directory path of all the files in the Output panel.
- Also, store the updated names in MS Word file and save and close it.

Build a workflow using **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 MS Word file and save and close it.

Algorithm

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

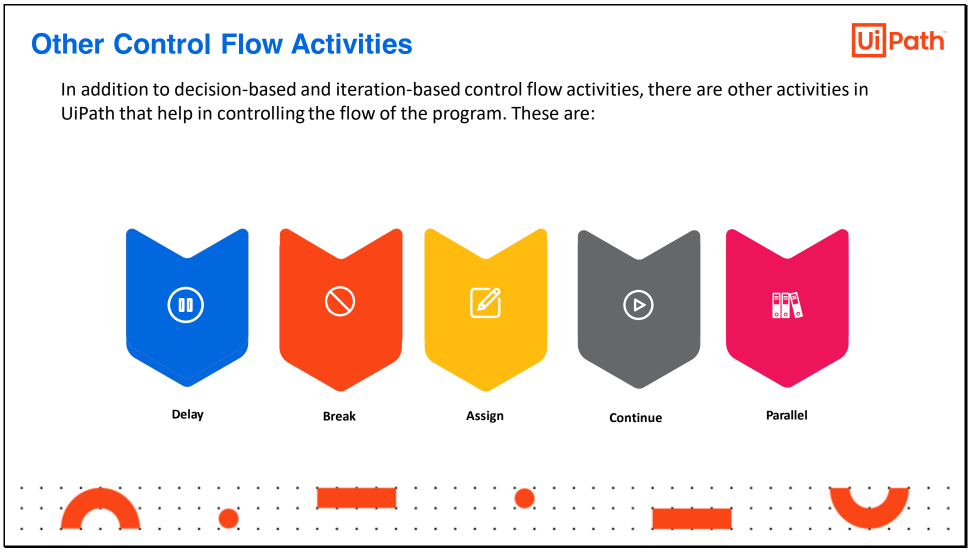


Other Control Flow Activities

- Delay
- Break
- Assign
- Continue
- Parallel

UiPath

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



Other Control Flow Activities

UiPath

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

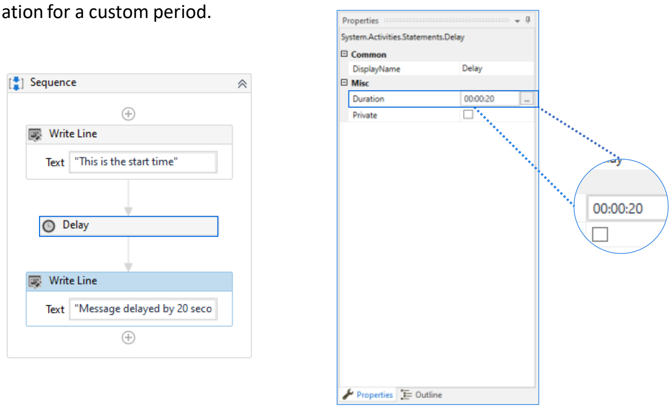
Parallel

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

Delay

Pauses the automation for a custom period.



Write Line
Text "This is the start time"

Delay

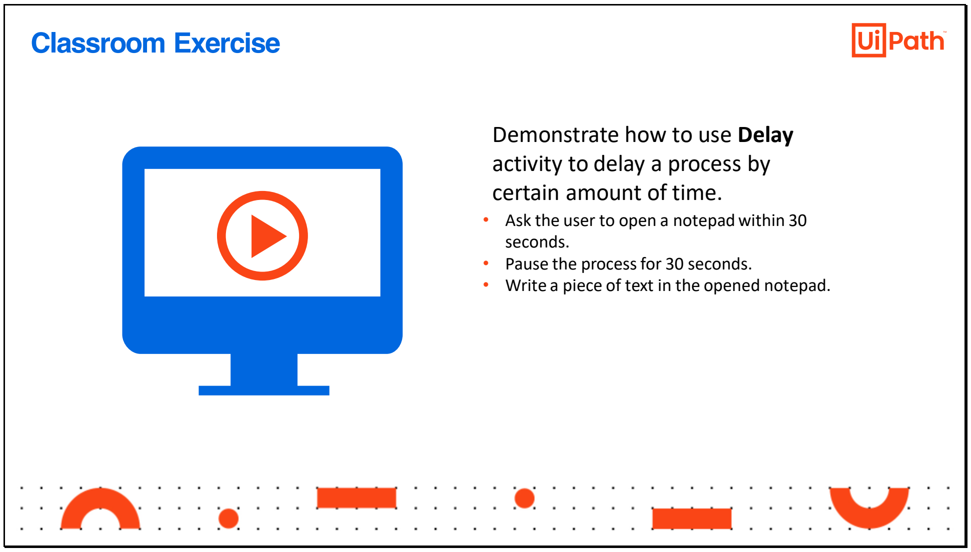
Write Line
Text "Message delayed by 20 seco

DisplayName Delay
Duration 00:00:20
Private □

The Delay activity enables the user to pause the automation for a custom period 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.



Classroom Exercise

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

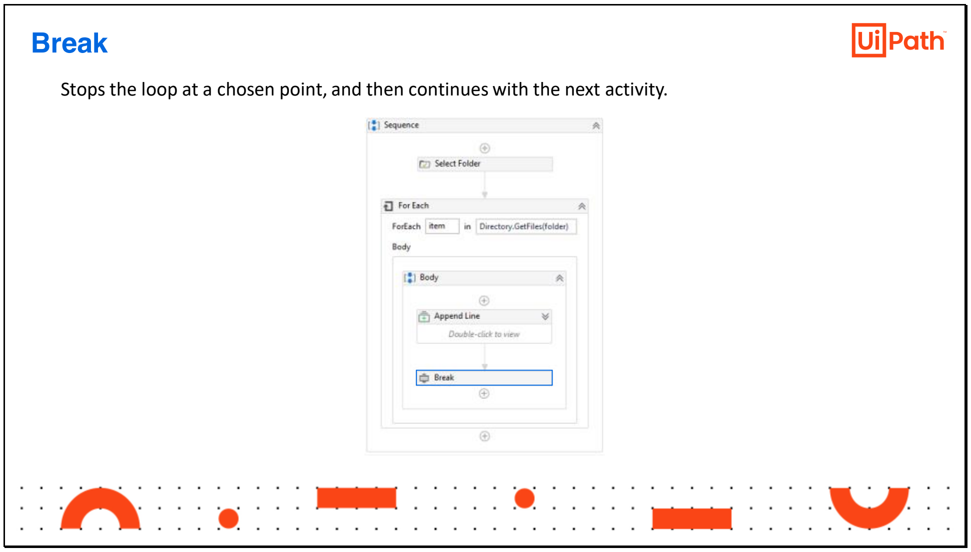
- Ask the user to open a notepad within 30 seconds.
- Pause the process for 30 seconds.
- Write a piece of text in the opened notepad.

Demonstrate how to use **Delay** activity to delay a process by 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 the UiPath Studio, insert a **Message Box** activity in the designer panel.
- Enter text "Hi, open a new notepad within 30 seconds."
- Insert a Delay activity below Message Box activity.
- Go to the Properties panel of the Delay activity and set duration as 30 seconds.
- Insert Type Into activity below Delay activity.
- Click "Indicate on screen" link and indicate the editor area of the already open notepad. Enter 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 Delay activity works. It makes the process wait for certain amount of time, and then continues running.

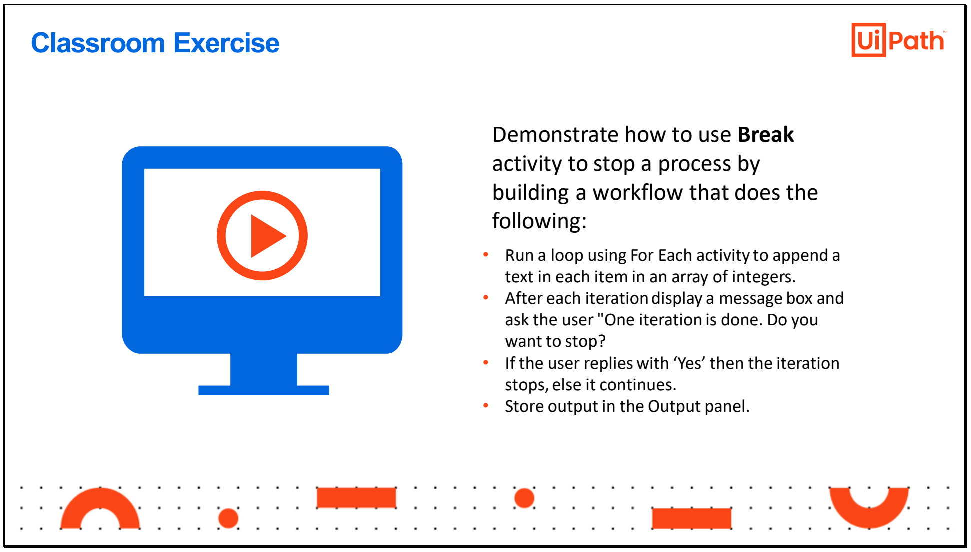


Break

Stops the loop at a chosen point, and then continues with the next activity.

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

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



Classroom Exercise

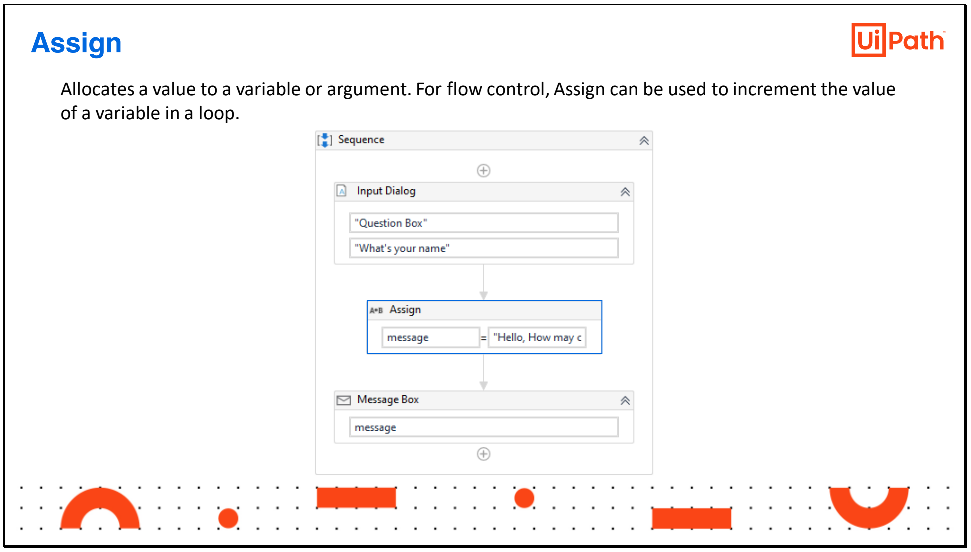
Demonstrate how to use **Break** activity to stop a process by building a workflow that does the following:

- Run a loop using 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.
- Store output in the Output panel.

Demonstrate how to use **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 **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 **For Each** activity below 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 **Input Dialog** activity below **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 **Variables** panel. Set Variable Type as **string**.
- Enter this variable in **Result** property of this activity using its **Properties** panel.
- Insert **If** activity below Input Dialog activity. Enter condition **userReply= "Yes"**. Insert a **Break** activity in the **Then** box.
- Run the program with few variations.



Assign

Allocates a value to a variable or argument. For flow control, Assign can be used to increment the value of a variable in a loop.

Sequence

Input Dialog

"Question Box"

"What's your name"

Assign

message = "Hello, How may c

Message Box

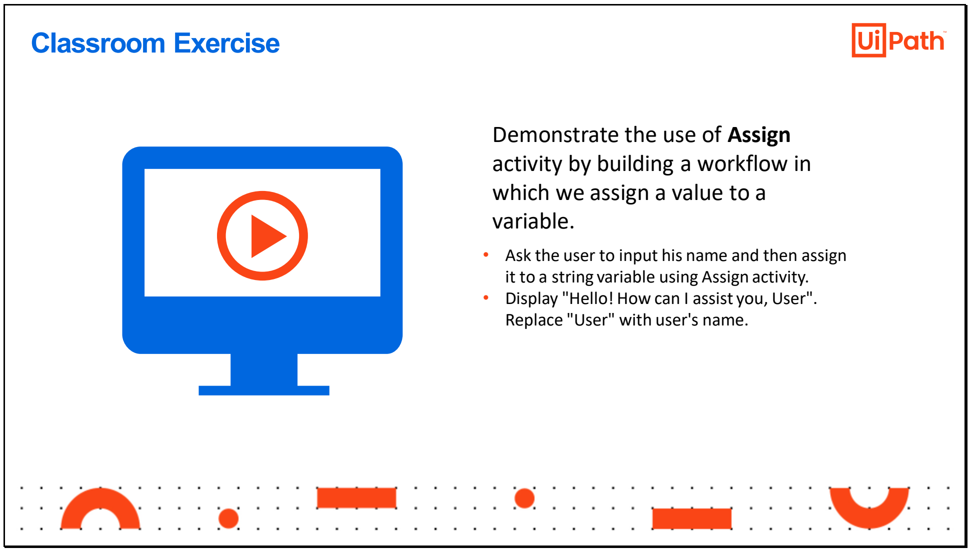
message

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 activity inputs a user's name and assigns it to a variable.



Classroom Exercise

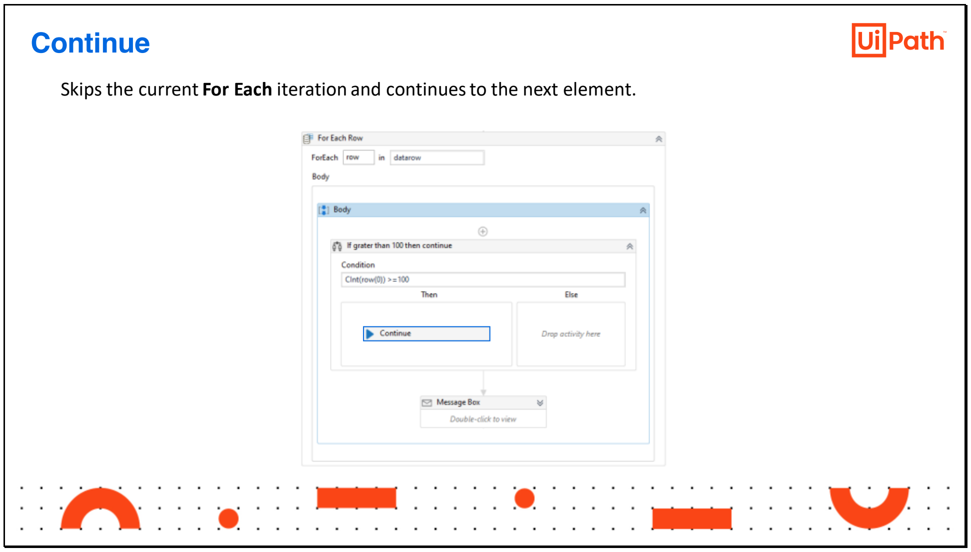
Demonstrate the use of **Assign** activity by building a workflow in which we assign a value to a variable.

- 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.

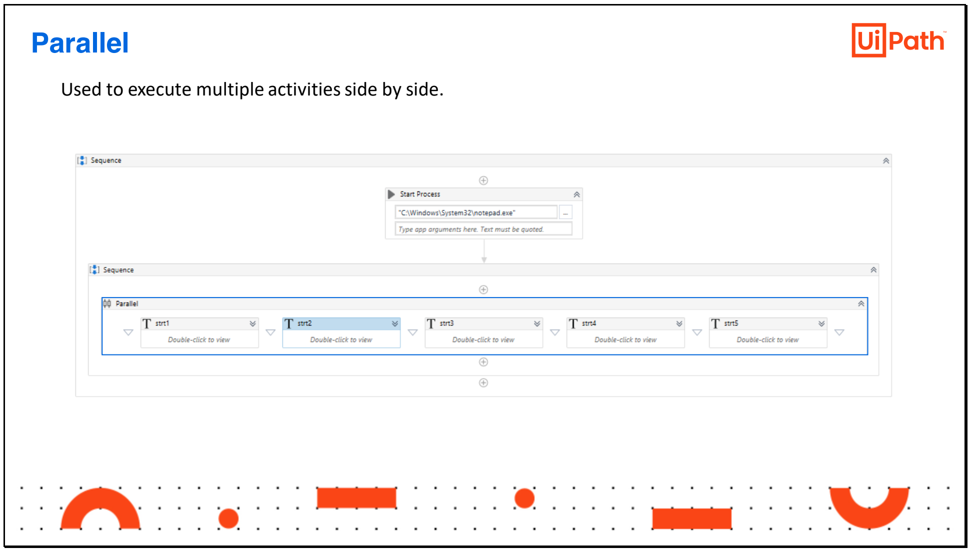


Continue

Skips the current **For Each** iteration and continues to the next element.

The Continue activity skips the current **For Each** iteration and continues to the next element.

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



Parallel

Used to execute multiple activities side by side.

"C:\Windows\System32\notepad.exe"

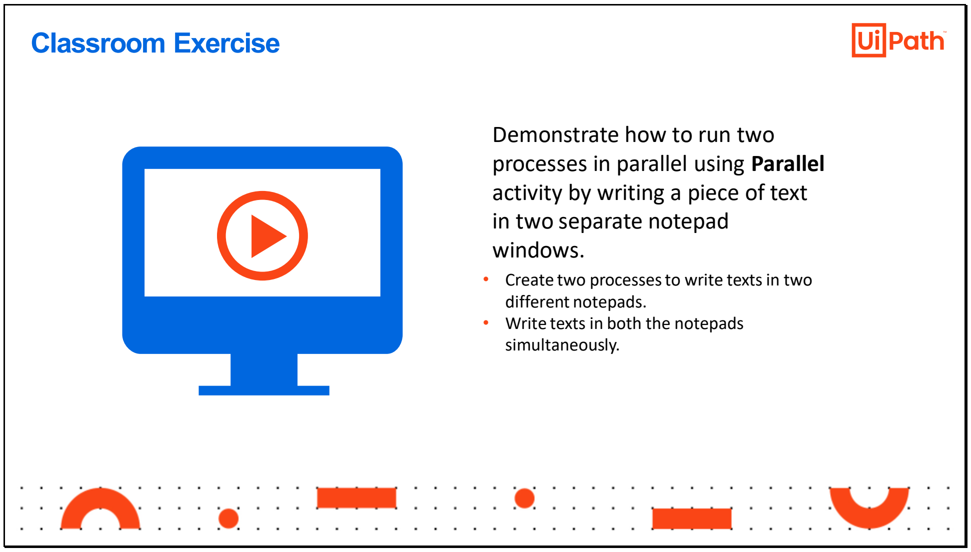
Type app arguments here. Text must be quoted.

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.

Parallel activity lets the user schedule two or more child activity branches for processing simultaneously. However, it cannot run multiple processes in foreground at the same time. In order to run more than one process at the same time, user has to choose a single activity branch to run in the foreground, and 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.



Classroom Exercise

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

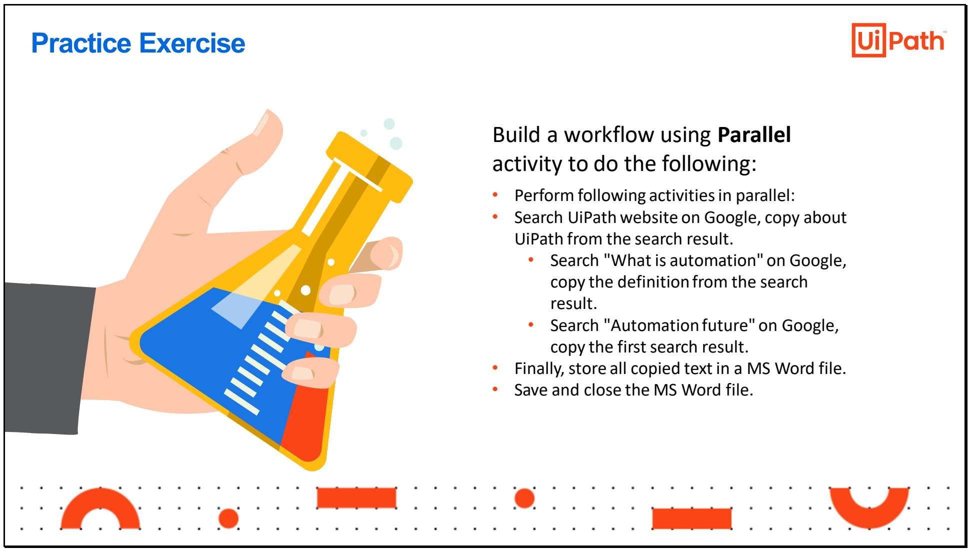
- Create two notepads with names “foreground.txt”, and “background.txt”, and open them side by side on the desktop.
- In the UiPath Studio, insert **Parallel** activity in designer panel, and insert two **Sequence** activities in it.
- Insert **Type Into** activities in both the **Sequence** activities.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

- Click **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.



Practice Exercise

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

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

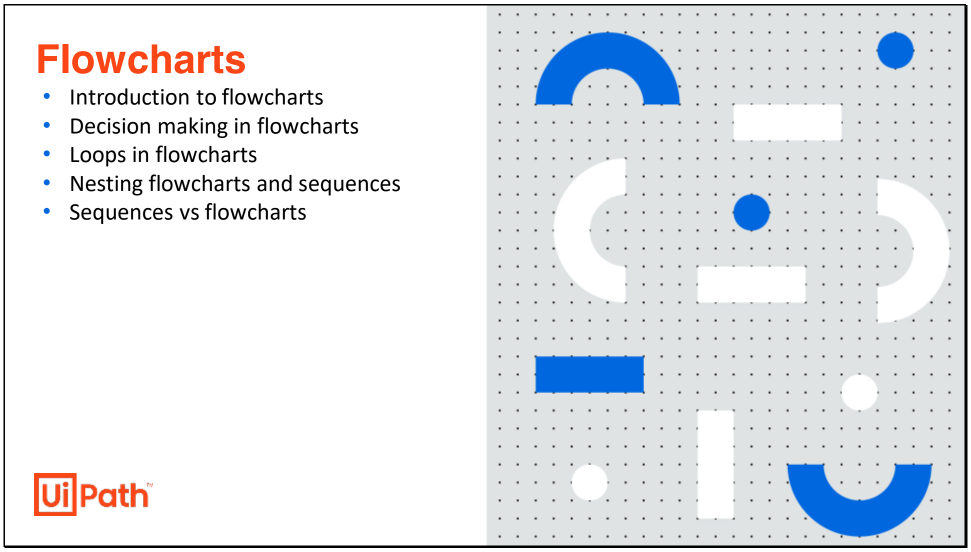
Build a workflow using **Parallel** activity which performs the following parallelly:

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

Algorithm

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

  - Set SendWindowMessages property to True of this Type Into activity.
  - Copy the search result using Get Text activity.
- In the third Sequence activity:
  - Use Open Browser activity to open Google website “www.google.com”
  - Use Type Into activity to search for “Automation is Future”.
  - Set SendWindowMessages property to True of this Type Into activity
  - Copy the search result using Get Text activity.
- Use a Type Into activity to insert text from all three parallel Sequence activities into a MS Word file.
- Step 22: Use Send Hotkey activities to Save and close the MS Word file.
- STOP

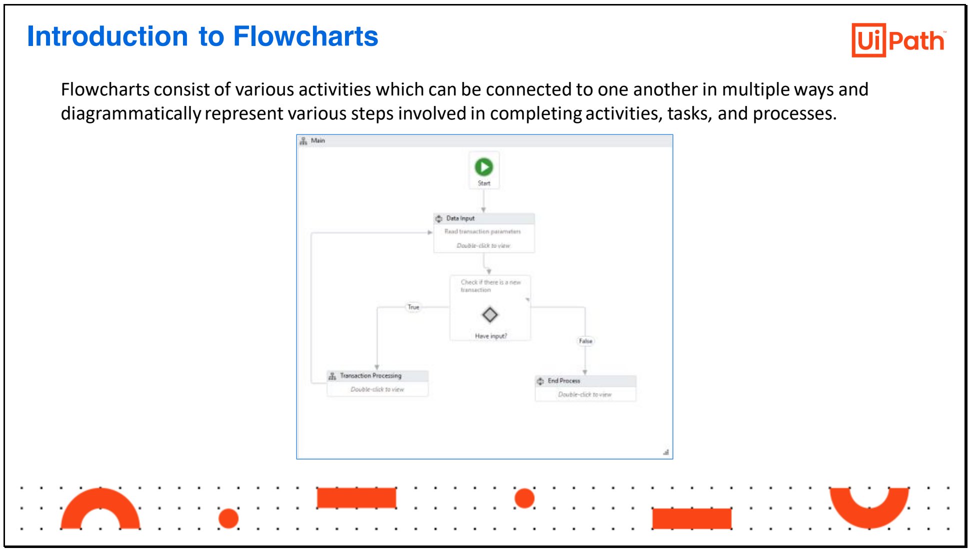


Flowcharts

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

UiPath™

This section explains flowcharts in detail.



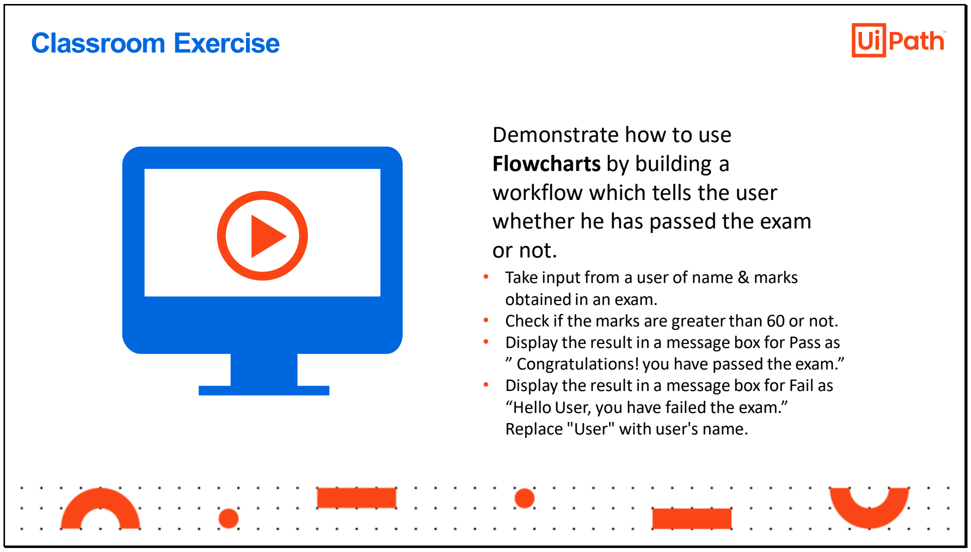
Introduction to Flowcharts

Flowcharts consist of various activities which can be connected to one another in multiple ways and diagrammatically represent various steps involved in completing activities, tasks, and processes.

A flowchart is a type of project that consists of various activities which can be connected to one another in multiple ways. It is diagrammatical approach which 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.



Classroom Exercise

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

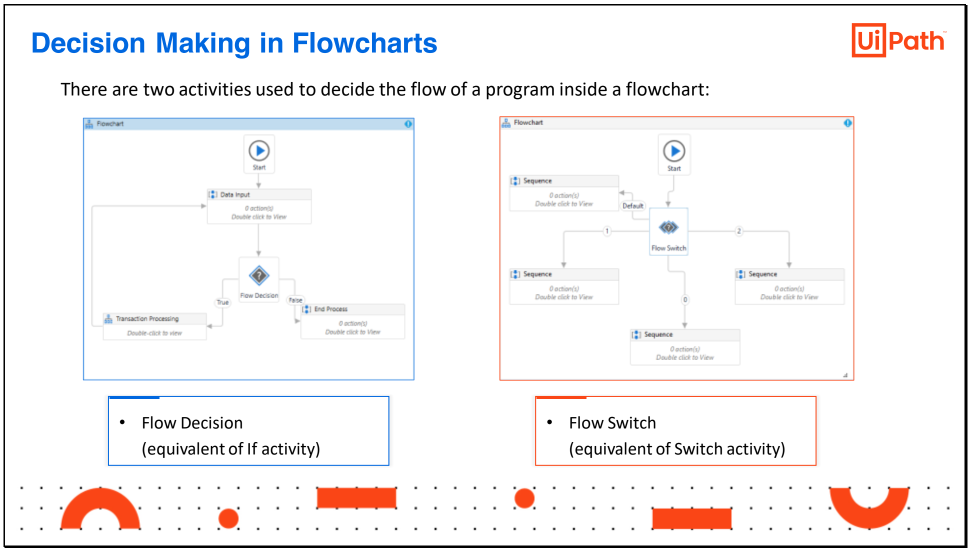
- Take input from a user of name & marks obtained in an exam.
- Check if the marks are greater than 60 or not.
- Display the result in a message box for Pass as " Congratulations! you have passed the exam."
- Display the result in a message box for Fail as "Hello User, you have failed the exam." Replace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

Demonstrate how to use **Flowcharts** by building a code which 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 **Variables** panel. Set Variable Type as **string**.
- Enter this variable in **Result** property of this activity using its **Properties** panel.
- Insert another **Input Dialog** activity below the first Input Dialog activity.
- Enter the text **"Marks"** in the *Title*, and **"Enter your Marks in numeric"** in the *Label*.
- Define a variable called **marks** for this activity using **Variables** panel. Set Variable Type as **Int32**.
- Enter this variable in **Result** property of this activity using its **Properties** panel.
- Insert **Flow Decision** activity below the second Input Dialog activity.
- Enter condition **marks >= 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 text **"Congratulations!" + names + ", you have passed the exam."**
- In the second Message activity, enter 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.



Decision Making in Flowcharts

There are two activities used to decide the flow of a program inside a flowchart:

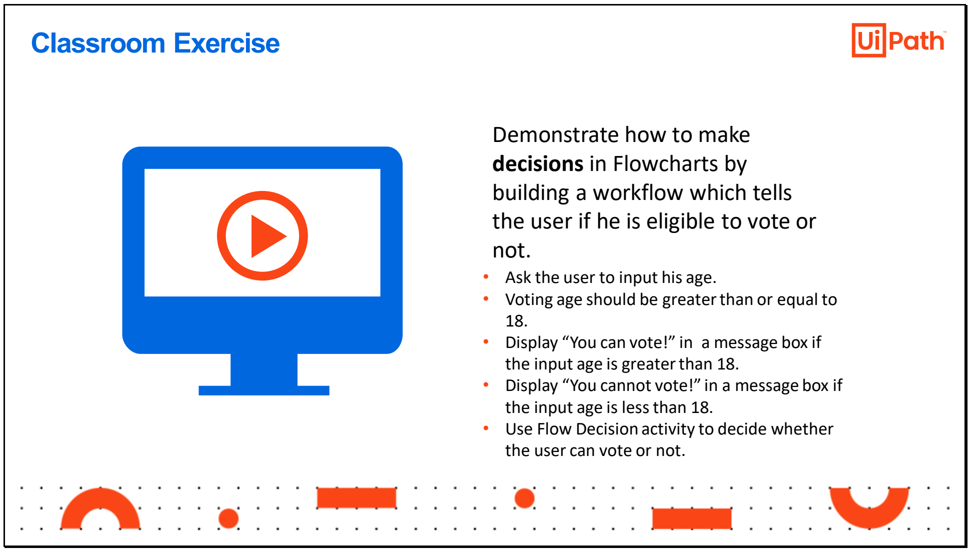
- Flow Decision
  (equivalent of If activity)

- Flow Switch
  (equivalent of Switch activity)

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

- **Flow Decision**: It is an activity which 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.



Classroom Exercise

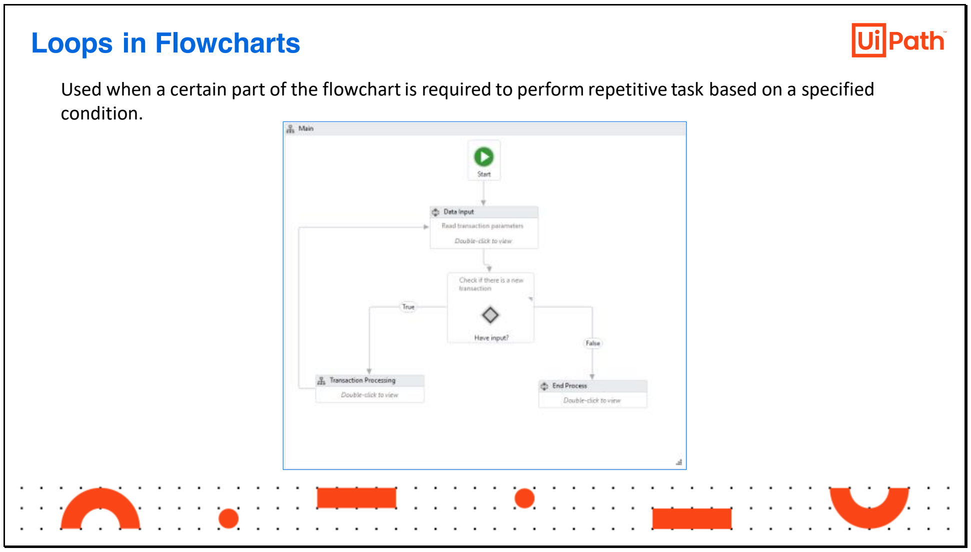
UiPath

Demonstrate how to make **decisions** in Flowcharts by building a workflow which 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 Flow Decision activity to decide whether the user can vote or not.

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

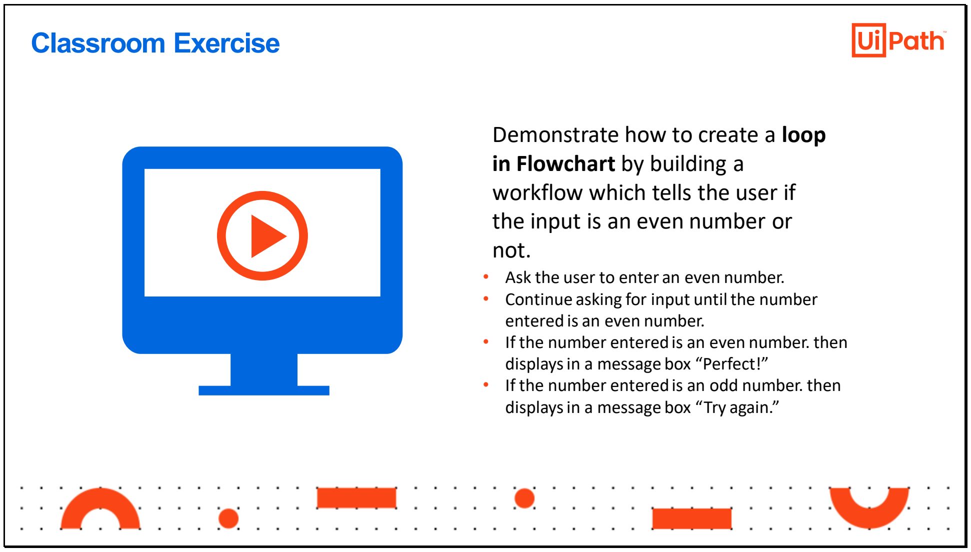
- Insert a Flowchart activity in the Designer panel.
- Insert Input Dialog activity below Start node.
- Enter text "Vote Eligibility" in the *Title*, and "Enter you age:" in the *Label* text area.
- Create a variable **userAge** with Variable type as **Int32** and Scope as **Flowchart**. Assign this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



Loops in Flowcharts

Used when a certain part of the flowchart is required to perform 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.



Classroom Exercise

UiPath

Demonstrate how to create a **loop in Flowchart** by building a workflow which 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 displays in a message box "Perfect!"
- If the number entered is an odd number. then displays in a message box "Try again."

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

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 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"

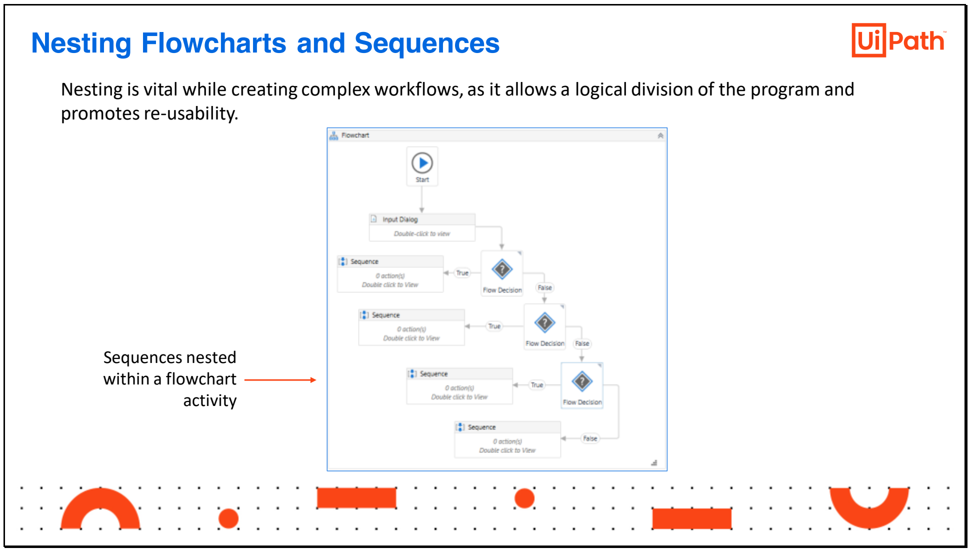
Build a workflow that asks user for his name and two-digit lottery number and displays if he is a winner.

- 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 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"

Algorithm

- START
- Use an Input Dialog activity within a Flowchart activity to get name of the user
- Use another Input Dialog activity to take 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 input is above 54.
    - If input is below 54, use Assign activity to increment count of chances used by 1
    - If input is above 54, use another Flow Decision to check if input is below 64
        - If input is above 64, use Assign activity to increment count of chances used by 1

        - If 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 count of chances used by the user reaches five.
    - Use a Message Box activity to display "Sorry, you lost. No more chances remaining"
- STOP

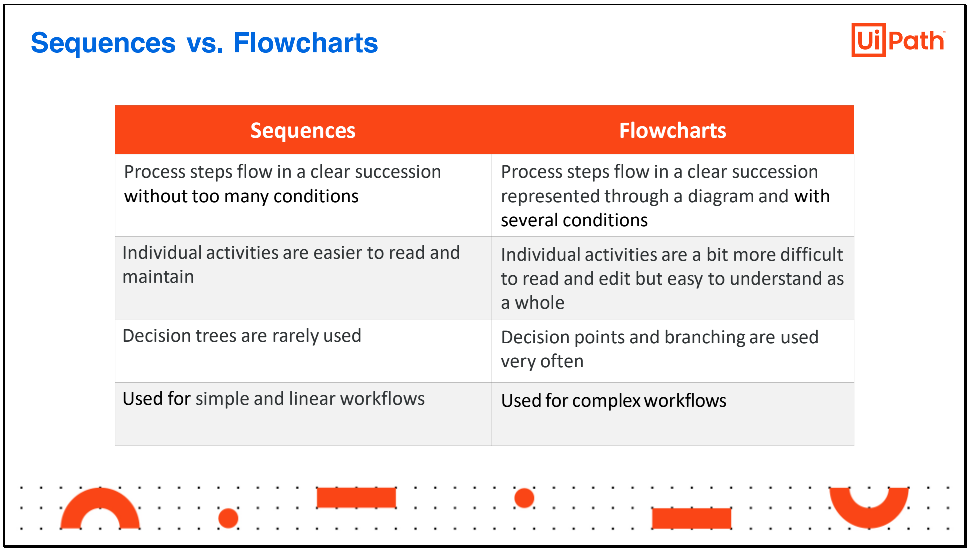


Nesting Flowcharts and Sequences

Nesting is vital while creating complex workflows, as it allows a logical division of the program and promotes re-usability.

Sequences nested within a flowchart activity

Nesting is vital while creating complex workflows, as it allows a logical division of the program and promotes re-usability. 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.

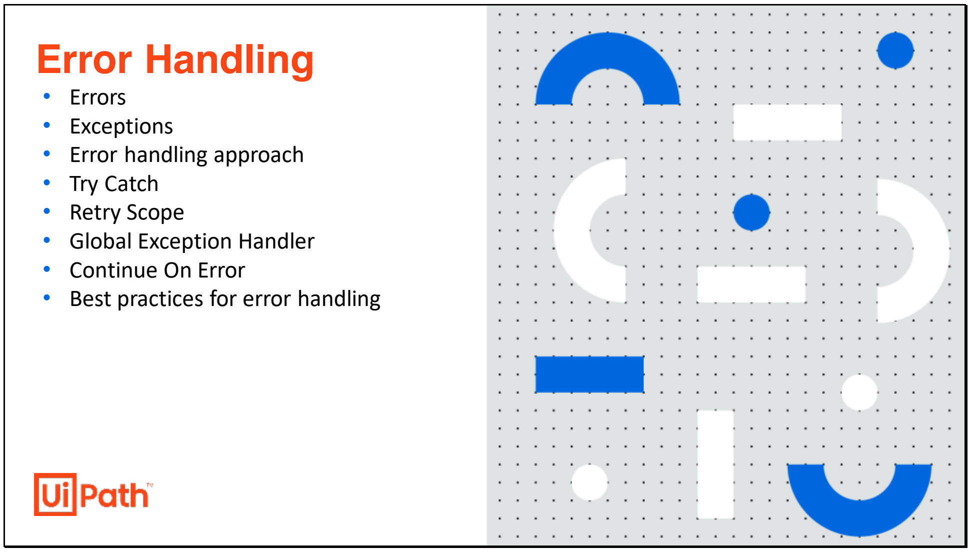


Sequences vs. Flowcharts

UiPath

Sequences	Flowcharts
Process steps flow in a clear succession without too many conditions	Process steps flow in a clear succession represented through a diagram and with several conditions
Individual activities are easier to read and maintain	Individual activities are a bit more difficult to read and edit but easy to understand as a whole
Decision trees are rarely used	Decision points and branching are used very often
Used for simple and linear workflows	Used for complex workflows

- In **Sequences**, the process steps flow in a clear succession. Decision trees are rarely used. Activities in sequences are easier to read and maintain, thus, they are highly recommended for simple, linear workflows.
- In **Flowcharts**, the individual activities are a bit more difficult to read and edit, but the flows between them are much clearer. Use flowcharts when decision points and branching are needed in order to accommodate complex scenarios, workarounds and decision mechanisms.

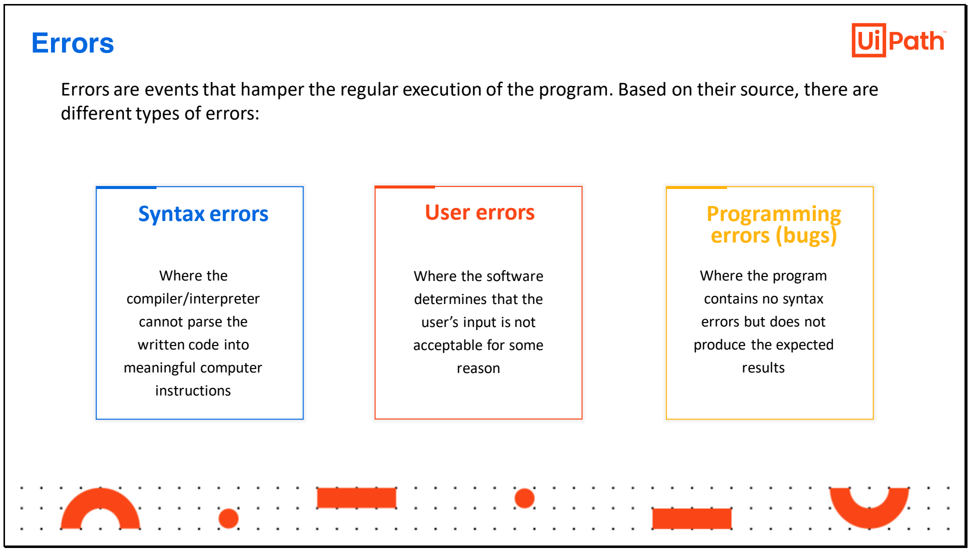


Error Handling

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

UiPath

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



Errors

UiPath

Errors are events that hamper the regular execution of the program. 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

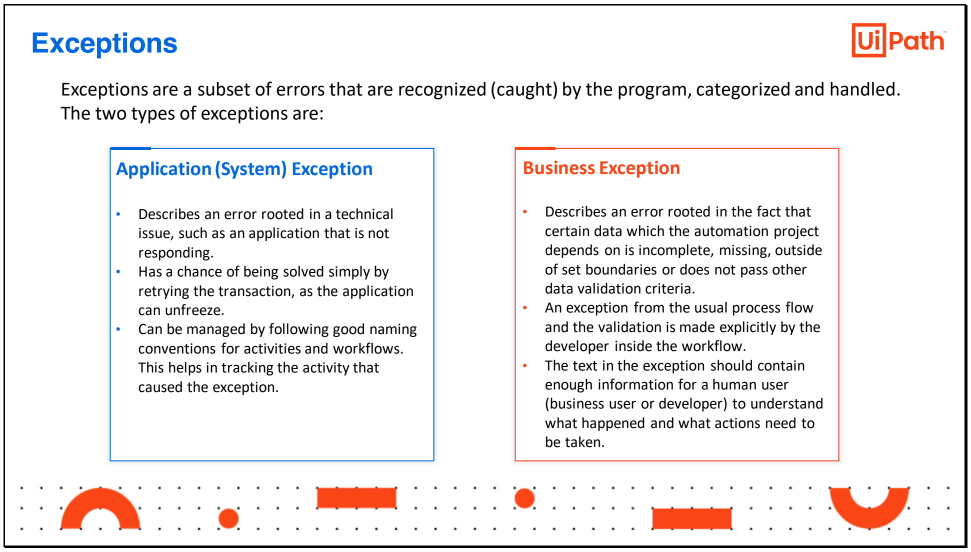
Programming errors (bugs)

Where the program contains no syntax errors but does not produce the expected results

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. This are often called bugs.



Exceptions

UiPath

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

- 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.
- 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.

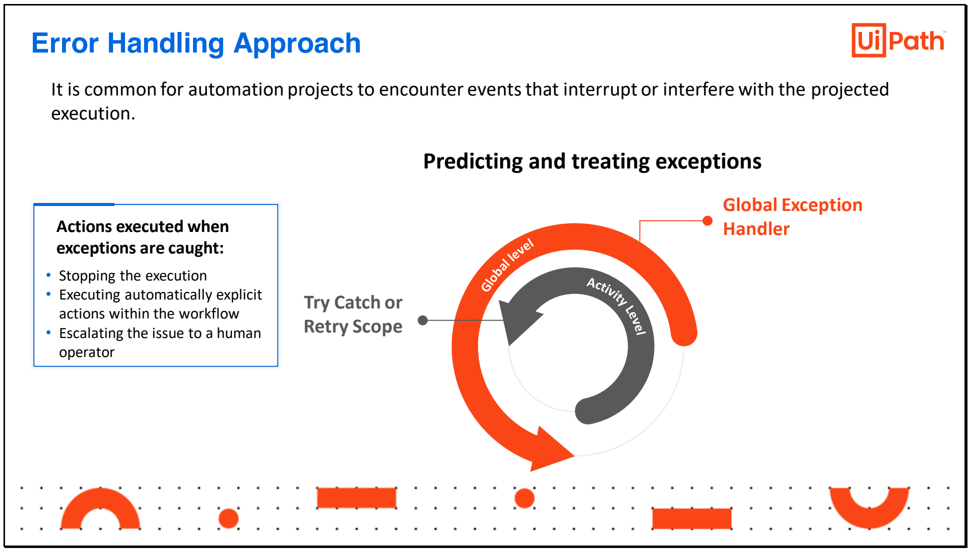
Exceptions are a subset of errors that are serious enough to produce a material effect and for which there can be a mechanism to identify and address them. Exception handling mechanism refers to how to prevent and/or respond to exceptions. Sometimes, the handling mechanism can be simply stopping the 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.



Error Handling Approach

UiPath

It is common for automation projects to encounter events that interrupt or interfere with the projected execution.

Predicting and treating exceptions

Actions executed when exceptions are caught:

- Stopping the execution
- Executing automatically explicit actions within the workflow
- Escalating the issue to a human operator

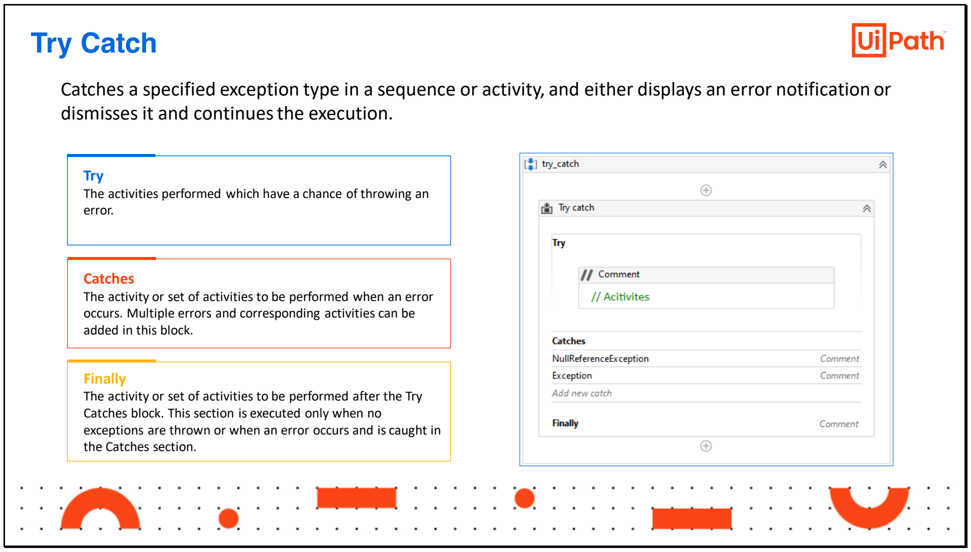
It is common for automation projects to encounter events that interrupt or interfere with the projected execution.

A good project design will include ways of recognizing and categorizing the exception, and patterns of action that are executed only when exceptions are caught:

- Simply stopping the execution
- Executing automatically explicit actions within the workflow
- Escalating the issue to a human operator

Predicting and treating exceptions can be done in two ways:

- **At activity level**: Using Try/Catch or Retry Scope;
- **At a global level**: Using the Global Exception Handler.



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.

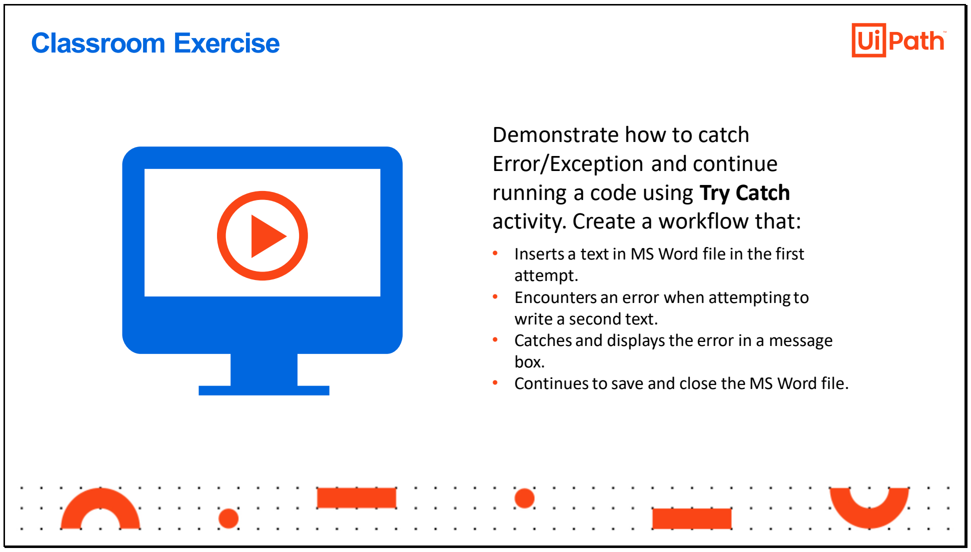
Finally
The activity or set of activities to be performed after the Try Catches block. This section is executed only when no exceptions are thrown or when an error occurs and is caught in the Catches section.

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 - The activity or set of activities to be performed after the Try and Catches blocks are executed. This section is executed only when no exceptions are thrown or when an error occurs and is caught in the Catches section.



Classroom Exercise

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

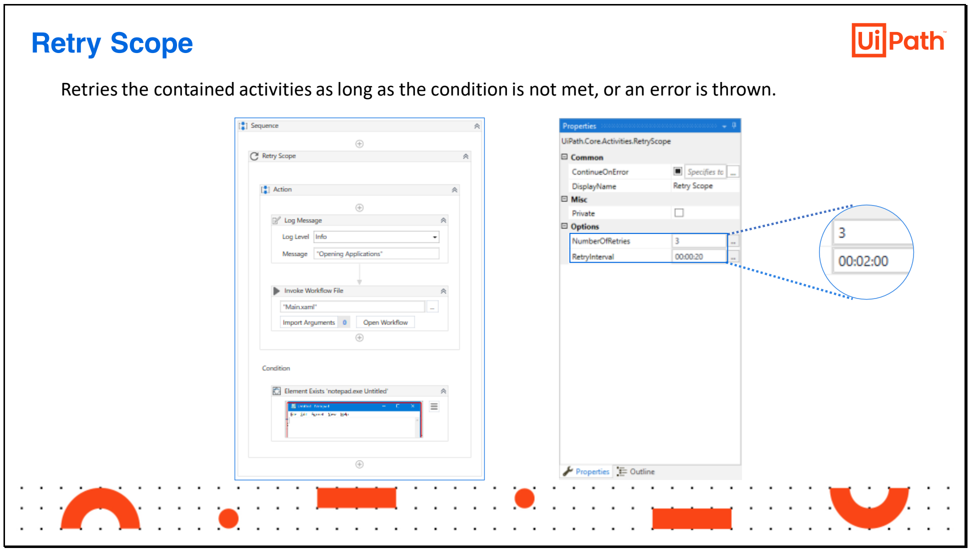
- Inserts a text in MS Word file in the first attempt.
- Encounters an error when attempting to write a second text.
- Catches and displays the error in a message box.
- Continues to save and close the MS Word file.

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

- Open MS Word application on the desktop called test.docx.
- Insert **Assign** activity in the designer panel.
- Click "Indicate on screen" link of **Type Into** activity and indicate the editor area of MS Word file.
- In the text area of Type Into activity, insert the text "Automation is the future".
- Insert **Try Catch** activity below the **Assign** activity.
- Insert **Type Into** activity within **Try** section of **Try Catch** activity.
- Click "Indicate on screen" link of **Type Into** activity and indicate the editor area of MS Word file.
- In the text area of **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 **Try Catch activity**.
- Insert a **Message Box** within **Choose** section. Insert text **exception.Message** in the text area.
- In the **Finally** section, insert two **Click** activities.
- Indicate **Save** button of the MS Word file with the first click activity, and **Close** button with the second click activity.
- Let's run the program...
    - First text gets written in the MS Word.
    - Process trying to write second text...but error message box shows up...click **OK**.
    - Process now clicks Save button and Close button of 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 program runs further and closes the MS Word application. It means our program did not stop even after facing error.



Retry Scope

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

Log Level: Info

Message: "Opening Applications"

"Main.xaml"

UiPath.Core.Activities.RetryScope

Common

ContinueOnError: ■ Specifies tc

DisplayName: Retry Scope

Misc

Private: □

Options

NumberOfRetries: 3

RetryInterval: 00:00:20

3

00:02:00

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

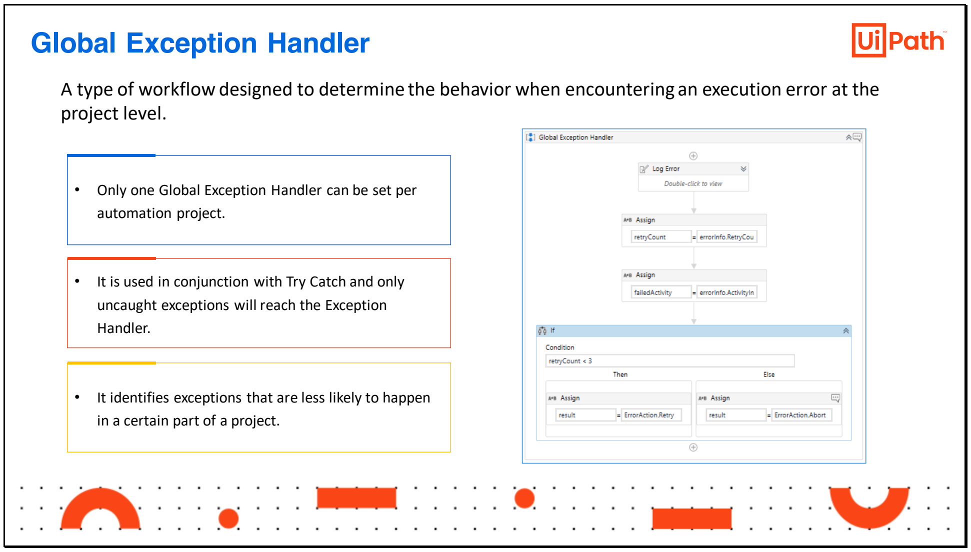
It is used to retry the execution in situations in which an error is expected. The execution will be retried until a certain event happens (for a number of times) or without any condition (retried until no exception is thrown).

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.



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 identifies exceptions that are less likely to happen in a certain part of a project.

Log Error: Double-click to view

Assign: retryCount = errorInfo.RetryCou

Assign: failedActivity = errorInfo.ActivityIn

If: Condition retryCount < 3

Then: Assign result = ErrorAction.Retry

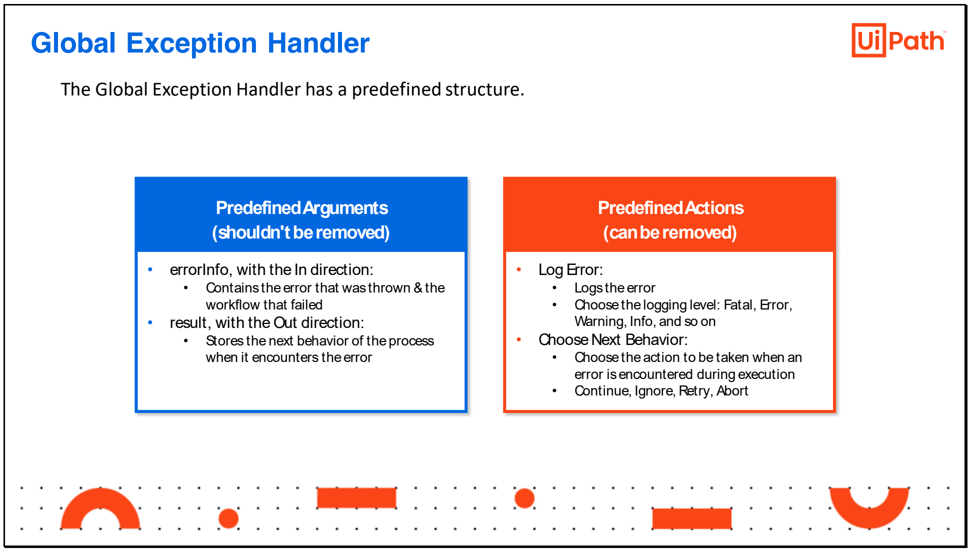
Else: Assign result = ErrorAction.Abort

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.

Its purpose is to identify exceptions that are less likely to happen in a certain part of a project.

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.



Global Exception Handler

UiPath

The Global Exception Handler has a predefined structure.

Predefined Arguments (shouldn't be removed)

- errorInfo, with the In direction:
  - Contains the error that was thrown & the workflow that failed
- result, with the Out direction:
  - Stores the next behavior of the process when it encounters the error

Predefined Actions (can be removed)

- Log Error:
  - Logs the error
  - Choose the logging level: Fatal, Error, Warning, Info, and so on
- Choose Next Behavior:
  - Choose the action to be taken when an error is encountered during execution
  - Continue, Ignore, Retry, Abort

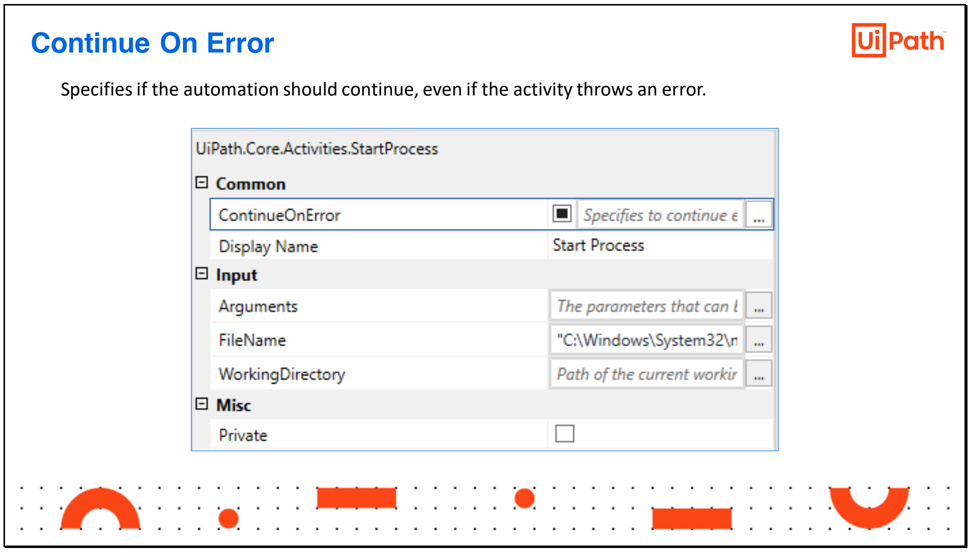
The Global Exception Handler has a predefined structure.

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



Continue On Error

UiPath

Specifies if the automation should continue, even if the activity throws an error.

UiPath.Core.Activities.StartProcess

⊟ **Common**	
ContinueOnError	■ *Specifies to continue e* ...
Display Name	Start Process
⊟ **Input**	
Arguments	*The parameters that can l* ...
FileName	"C:\Windows\System32\n ...
WorkingDirectory	*Path of the current workir* ...
⊟ **Misc**	
Private	□

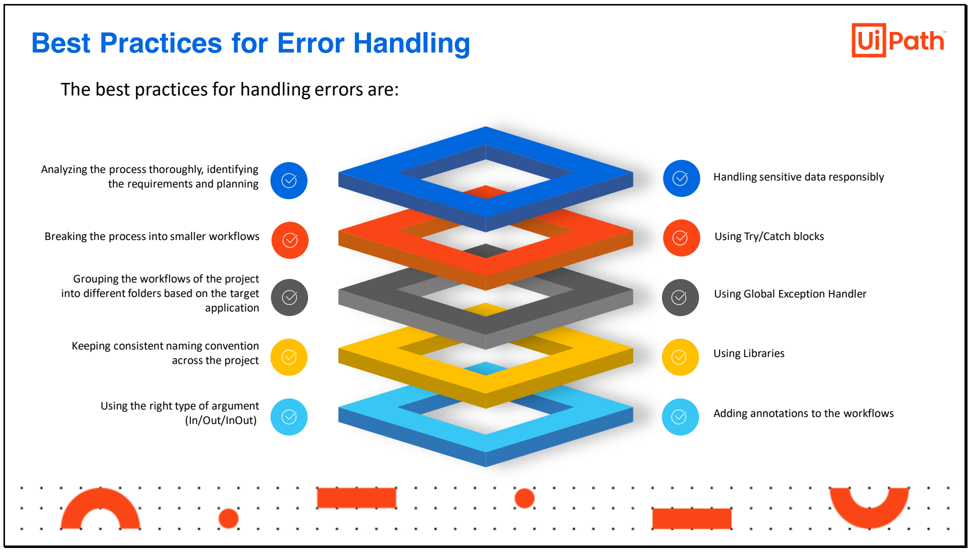
Continue On Error is very useful for activities that work with UI interactions. It is the easiest way to tell the workflow to continue even if the activity throws an error.

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.



Best Practices for Error Handling

UiPath

The best practices for handling errors are:

Analyzing the process thoroughly, identifying the requirements and planning

Breaking the process into smaller workflows

Grouping the workflows of the project into different folders based on the target application

Keeping consistent naming convention across the project

Using the right type of argument (In/Out/InOut)

Handling sensitive data responsibly

Using Try/Catch blocks

Using Global Exception Handler

Using Libraries

Adding annotations to the workflows

The best practices for handling errors are:

- Analyzing the process thoroughly, identifying the requirements and planning how the solution should look like before starting the actual development.
- Breaking the process into smaller workflows for a better understanding of the code, independent testing and reusability.
- Grouping the workflows of the project into different folders based on the target application.
- Keeping a consistent naming convention across the project.
- Using the right type of argument (In/Out/InOut) when invoking a workflow.
- Handling sensitive data responsibly: no credentials should be stored in the workflow directly.
- Using Try/Catch blocks to predict and handle exceptions.
- Using Global Exception Handler for situations that are global and/or less probable to happen.
- Using Libraries for creating and storing reusable components for the projects.
- Adding annotations to the workflows to clarify the purpose of each workflow.

The users should always use logs in production to get relevant information.

Practice Exercise

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

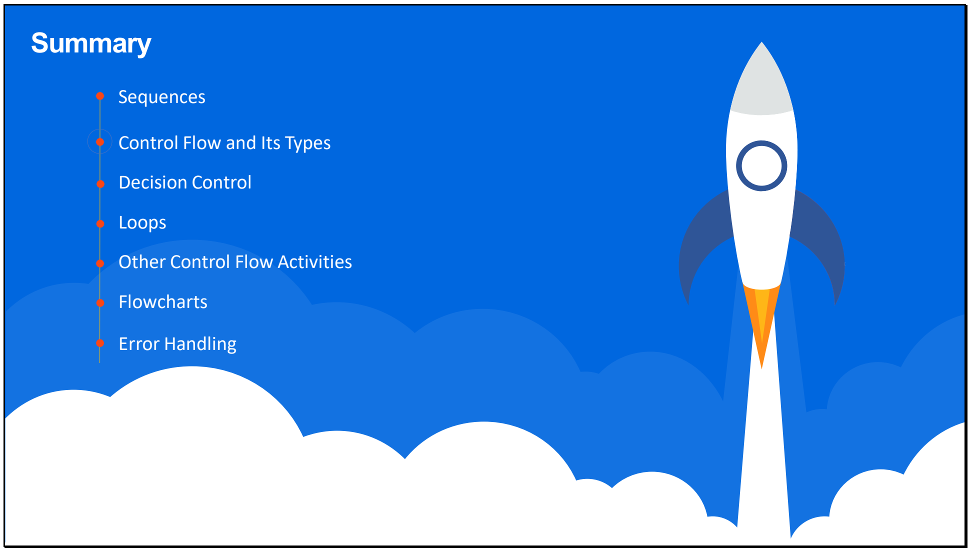
- Take Name, Gender, and Age as user input.
- Subtract current year with Age value to get Year of Birth.
- Handle an error that occurs due to a reckless user input of a wrong age containing 11-digit number.
- Continue process to record Name, Gender, and Year of Birth of the user in a Notepad.

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

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

Algorithm

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



Summary

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

To summarize, this lesson explained:

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